TCP/IP-gestützte Kopplung Flash-basierter GUIs an die Matlab/Simulink-Toolkette zur Visualisierung, Animation und Bedienung von parallelen, verteilten und internetfähigen Simulationen

H. Voß, G. Karrasch
voss@tfh-bochum.de; karrasch@tfh-bochum.de
Technische Fachhochschule Georg Agricola zu Bochum, Fachbereiche Maschinen- und Verfahrenstechnik, Elektro- und Informationstechnik
Herner Str.45, 44787, Bochum

Kurzfassung

Um professionellen Anforderungen gerecht werden, wird in diesem Beitrag eine Lösung zur Anbindung einer GUI an die Matlab / Simulink-Toolkette vorgeschlagen, deren GUI-Frontend auf dem Industrie-Standard Flash aufsetzt.

Die Kommunikation zwischen der Flash-basierten GUI und der Matlab-Komponente erfolgt über permanente bidirektionale Socket-Verbindungen, die über einen in Java implementierten Multithreaded-TCP/IP-Server (Socket Server) aufgebaut und verwaltet werden. Flash-GUI und Matlab fungieren dabei als Socket-Clients.

Die bidirektionale Kommunikation erfolgt über den Austausch von XML-Nachrichten. Verteilte und internet-fähige Simulationsanwendungen mit sehr kleinen Latenzzeiten sowie der Parallelbetrieb von Clientapplikationen sind möglich.

1 Einleitung

Ein wiederkehrendes Problem beim Einsatz von modellbasierten Simulationen besteht darin, für die Bedienung und für die Visualisierung sowie Animation von Simulationsergebnissen geeignete "Graphical User Interfaces" (GUIs) zu entwickeln, die professionellen Anforderungen gerecht werden, wie z. B. :

- Verwendung von Industrie - Standards und offenen Plattformen.
- Durchführbarkeit von parallelen, verteilten Simulationen in Echtzeit mit echtzeitnahen Interaktionen.
- Einfache Umsetzung und Änderbarkeit des GUI - Layouts gemäß der Anforderungen an das Corporate Design,
- Erhöhung der Wiederverwendbarkeit durch Berücksichtigung des objektorientierten Paradigmas bei der Entwicklung [1], [3], [4].

Da im zunehmenden Maße modellbasierte Simulationen auch für Präsentationszwecke vor Entscheidungsträgern potentieller Kunden eingesetzt werden, spielt die Professionalität der grafischen Gestaltung der GUI eine immer stärkere Rolle.

Für Software-In-the-Loop / Hardware-In-the-Loop (SIL/HIL) - Anwendungen in Matlab oder Matlab - / Simulink - xPC-Target [4], [5] bietet Matlab/Simulink die Möglichkeit, grafische Bedienoberflächen und Animationen unter Verwendung des Matlab-eigenen GUI-Generators und der VRML-Toolbox zu erstellen.

Beide besitzen jedoch Bezug nehmend auf die oben genannten Anforderungen Einschränkungen, insbesondere hinsichtlich der Flexibilität in der grafischen Gestaltung oder der Menü geführten Steuerung von Interaktionen mit dem Anwender.

Um die zu Beginn erwähnten Anforderungen zu erfüllen, wird in diesem Beitrag eine Lösung zur Anbindung einer GUI an die Matlab / Simulink-Toolkette vorgeschlagen, deren GUI-Frontend auf dem Industrie-Standard Flash [2] aufsetzt.

2 System-Architektur

Die Kommunikation zwischen der Flash-basierten GUI und der Matlab-Komponente erfolgt über permanente bidirektionale Socket-Verbindungen, die über einen in Java [1] implementierten Multithreaded-TCP/IP-Server aufgebaut und verwaltet werden.

Das nachfolgende UML-Verteilungsdiagramm (s.Abb. 1) zeigt eine mögliche HW- / SW – Verteilungsstruktur für die hier diskutierte Anwendung.

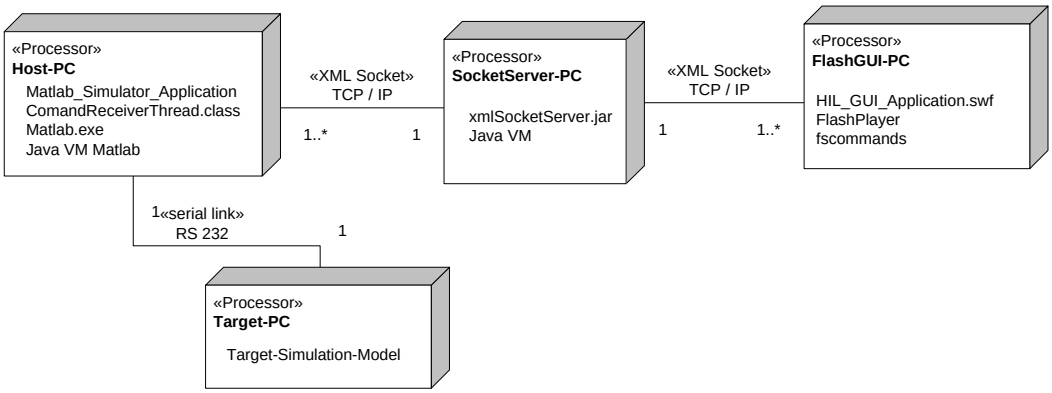

Abb. 1: HW - / SW - Verteilungsdiagramm

Auf dem Host-PC befinden sich zur Laufzeit neben der Matlab-Laufzeitumgebung (Matlab.exe), die eigentliche Matlab-Anwendung (Matlab_Simulator_Application), eine Java-basierte Anwendungsklasse (CommandReceiverThread.class), die zum Empfang der von der Flash-GUI (HIL_GUI_Application.swf) gesendeten XML-Nachrichten dient sowie die in Matlab integrierte Java-Laufzeitumgebung (Java VM Matlab).

Der SocketServer-PC enthält die Komponente xmlSocketServer.jar, die den Java-Bytecode der den Socket-Server realisierenden Klassen enthält sowie die hierfür notwendige Java-Laufzeitumgebung (Java VM).

Der FlashGUI – PC beinhaltet die Flash-basierte Bedienoberfläche (HIL_GUI_ Application.swf) sowie den für Flash-Anwendungen notwendigen Flash Player [7].

Beide, die Flash-GUI (HIL_GUI_Application.swf) und die Matlab-Anwendung (Matlab_Simulator_Application), fungieren in diesem Szenario als Socket-Clients, der TCP/IP-Server dagegen realisiert einen so genannten Socket-Server. Da die Socket-Verbindungen permanent geöffnet bleiben, sind echtzeitnahe Interaktionen mit bzw. zwischen den Modellsimulationen möglich.

Aus software - technischer Sicht betrachtet, existieren drei Subsysteme (s. Abb. 2):

- Socket-Server-Subsystem: beinhaltet alle zur Umsetzung eines Multithreaded-TCP/IP-Server notwendigen Java-basierten Klassen.
- Matlab-Subsystem: enthält alle Matlab-basierten Anwendungsklassen, die zur Implementierung des Simulationsmodells notwendig sind sowie Java-basierte Anwendungsklassen, die der Kommunikation mit dem SocketServer dienen.
- Flash-Subsystem: umfasst alle ActionScript–basierten Anwendungsklassen [6], die zur Realisierung der Flash-GUI erforderlich sind.

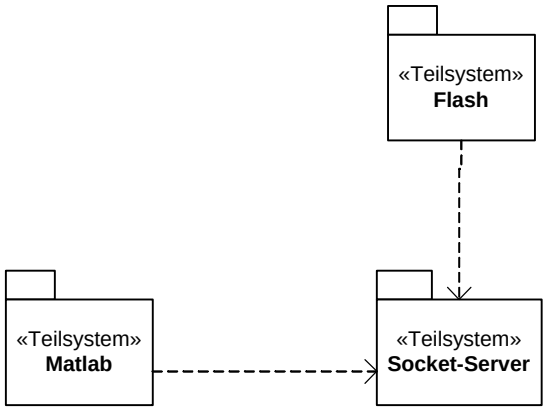

Abb. 2: Software-Subsystem-Struktur als UML – Klassendiagramm

3 Das Socket-Server-Subsystem

Das Socket-Server-Subsystem enthält alle Java-basierten Klassen, die zur Realisierung eines Multithreaded–TCP/IP-Server notwendig sind. Der Socket-Server etabliert und verwaltet zu den einzelnen Clients permanente bidirektionale Socket-Verbindungen, die mit Hilfe der Java-Klassen ServerSocket und Socket implementiert werden. Die zentrale Aufgabe des Socket-Servers besteht darin, von einem Client empfangene XML-Nachrichten als Zeichenstrom an alle andere mit ihm verbundenen Clients weiter zu leiten. Hierfür besitzt die den Socket-Server realisierende Klasse XMLSocketServer die Methode broadcast() (s. Abb. 3).

Die Struktur des Socket-Server-Subsystems ergibt sich aus dem nachfolgenden UML-Klassendiagramm (Abb. 3). Die Java-basierten Applikationsklassen sind hierin grau unterlegt, die restlichen Klassen sind von diesen benötigte Klassen aus dem Java API.

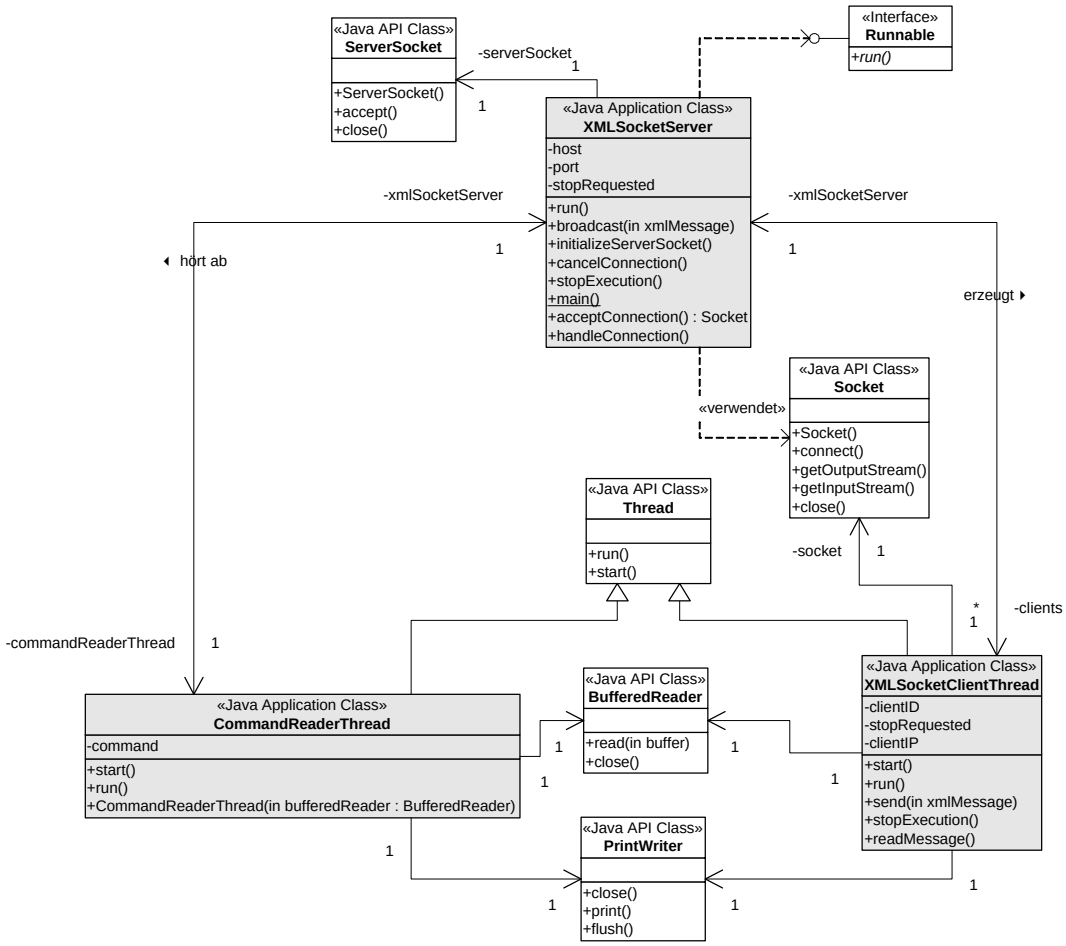

Abb. 3: UML – Klassendiagramm zum SocketServer-Subsystem

Die Java-Applikationklasse XMLSocketServer enthält den Programmeinstiegspunkt (main-Methode) der Anwendung und bildet zugleich den als Thread im Hintergrund ablaufenden Multithreaded–TCP/IP-Server.

Beim Aufruf der main-Methode von XMLSocketServer wird lediglich der Konstruktor von XMLSocketServer aufgerufen, der seinerseits zunächst das für den Server erforderliche ServerSocket-Objekt initialisiert (initializeServerSocket()) sowie verschiedene Eingabe- bzw. Ausgabeströme anlegt, anschließend sich selbst sowie ein Objekt der Klasse CommandReaderThread als unabhängige Threads startet. Die Klasse CommandReaderThread stellt während der Laufzeit ein einfaches Command Line Interface (CLI) zur Administration des TCP/IP-Servers zur Verfügung.

Nach dem Starten durchläuft der XMLSocketServer in seiner run-Methode eine Endlosschleife, innerhalb derer er durch die blockierende Methode acceptConnection() auf Anfragen von Clients wartet.

Zur Bearbeitung von Client-Anfragen startet der Multithreaded–TCP/IP-Server (XMLSocketServer) für jeden Client jeweils einen separaten Request-Handler (Objekt der Klasse XMLSocketClientThread) als eigenen Thread. Jedes Request-Handler-Objekt kommuniziert dabei über eine eigene Socket-Verbindung mit dem XMLSocketServer-Objekt. Die Socket-Verbindung wird über ein Socket-Objekt hergestellt, das die Methode acceptConnection() des XMLSocketServer-Objekts für den Client bereitstellt. Zu diesem Socket-Objekt gehören entsprechende InputStream - / OutputStream-Objekte über die die XML-Nachrichten mit Hilfe der Methoden readMessage() und send() ausgetauscht werden können.

Innerhalb seiner run-Methode wartet der Request-Handler (XMLSocketClientThread) auf XML-Nachrichten, die mit Hilfe der Methode readMessage()über die etablierte Socket-Verbindung eingelesen werden. Anschließend werden die empfangenenen Nachrichten durch Aufrufen der broadcast-Methode des XMLSocketServers an alle anderen Request-Handler-Objekte (XMLSocketClientThread-Objekte) gesendet.

4 Das Matlab-Subsystem

Das Klassendiagramm zum Matlab-Subsystem ist in der Abb. 4 dargestellt.
Um einen Daten- und Befehlstransfer zu Matlab/Simulink-Anwendungen zu realisieren, wird Matlab mit einer Socket-Schnittstelle (MatlabToSocketConnector im Matlab_Socket_IF-Paket) und einem spezifisch gepufferten Ein-/Ausgabe-Thread (CommandReceiverThread) ausgestattet. Da Matlab über ein Java-API verfügt, sind beide Komponenten ebenfalls in Java implementiert worden.

Die Pufferung dient zum einen der Formatanpassung (Dekodierung) der XML-Nachrichten, die von Flash über die XML-Socket-Verbindung an Matlab gesendet werden und zum anderen der verlustfreien asynchronen Kopplung zu Matlab-Routinen.

Diese Java-Schnittstelle beschränkt sich nicht allein auf die Verbindung eines Flash- und Matlab-Clients. Sie kann auch dazu genutzt werden, mehrere Matlab-Instanzen parallel zu betreiben und miteinander zu verbinden oder kann genauso gut in anderen Java-fähigen Clients verwendet werden.

Arbeiten mehrere Clients parallel, so können diese auf gleichen oder unterschiedlichen Rechnern im Netzwerk laufen. Jeder Client baut eine eigene permanente Socket-Verbindung mit dem zentralen Server auf und benutzt seinen eigenen Ein- / Ausgabe-Thread.

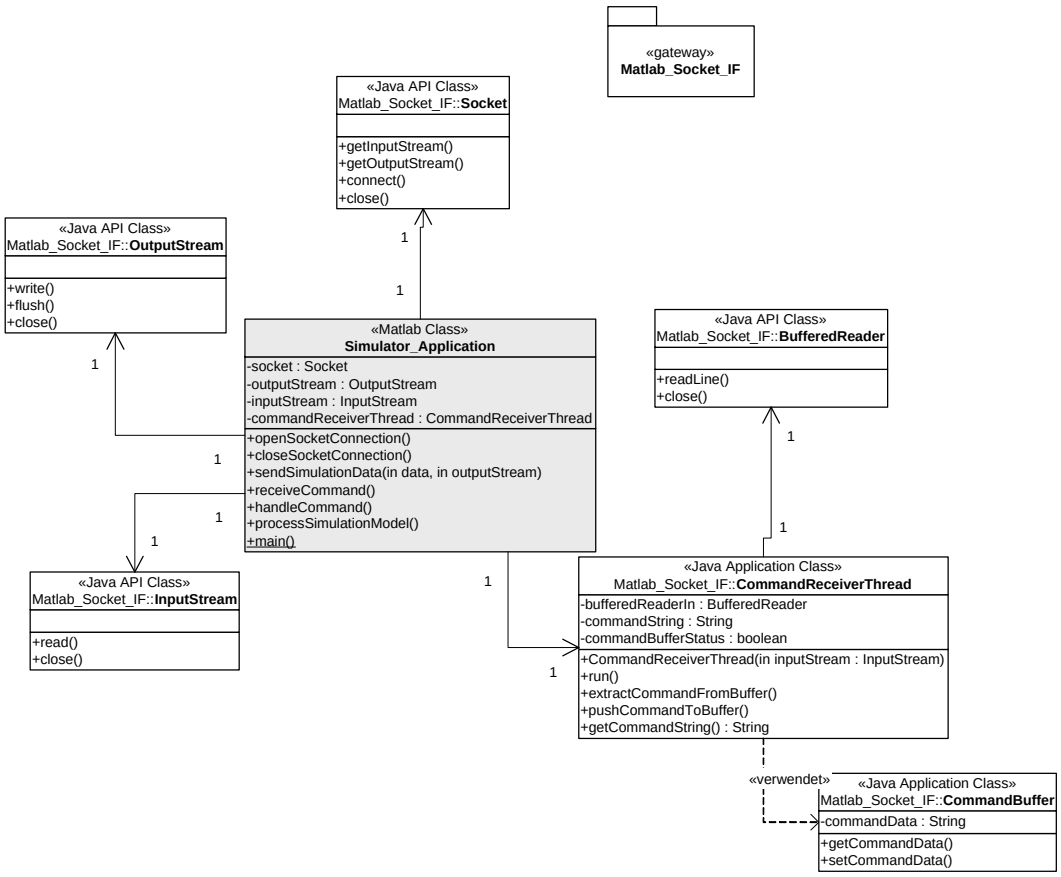

Abb. 4: UML – Klassendiagramm zum Matlab-Subsystem

Werden im Ein-/Ausgabe-Thread (CommandReceiverThread) eines Clients Sende-Strings beliebiger Clients empfangen, so sind diese zunächst im Empfangspuffer (BufferedReader) gespeichert. Über readLine() wird dieser Buffer ereignisgesteuert ausgelesen und zur weiteren Auswertung zwischengespeichert (pushCommandToBuffer) und dabei ggf. mit noch nicht ausgewerteten Strings verkettet.

Die Auswertung (extractCommandFromBuffer) erkennt und beseitigt die Null-Bytes der XML-Nachrichten und stellt diese im CommandBuffer dem Client-Programm zur Verfügung.

Beim Lesen aus dem CommandBuffer wird der Inhalt gelöscht. Um Verluste zu vermeiden, werden neu ausgewertete Strings solange an die vorhergehend gespeicherten Daten angehängt, bis der Zugriff erfolgte. Ein Flag (commandBufferStatus) signalisiert den Zustand des Speichers. Dieses wird vom Anwendungsprogramm (handleCommand in der Simulator_Application Klasse) ausgewertet.

Dies ist damit der Kopplungspunkt zur Anwendung, die in einer anderen Sprache, in unserem Falle in Matlab, programmiert sein kann, dafür allerdings eine Java-API benötigt.

Das Anwendungsprogramm prüft den Speicherinhalt, erkennt XML-Strukturen, die dann geparst werden. Andere Protokollformate werden anwendungsspezifisch gehandhabt.

Die Socket-Schnittstelle wird vom Anwendungsprogramm auch genutzt, um Daten an den Flash-Client zu übertragen (sendSimulationData).

5 Das Flash-Subsystem

Das Flash-Subsystem enthält alle ActionScript 2 – basierten Anwendungsklassen, die zur Realisierung der Flash-GUI erforderlich sind (s. Abb. 5).

Flash verwendet das sog. SWF-Dateiformat zur Beschreibung von interaktiven Animationen. SWF-Dateien enthalten dabei sowohl interpretierbaren Bytecode als auch Ressourcen wie Bilder und Sounds. Der ähnlich wie Java auf einer stack-orientierten virtuellen Maschine basierende Flash-Player interpretiert diesen Bytecode zur Laufzeit.

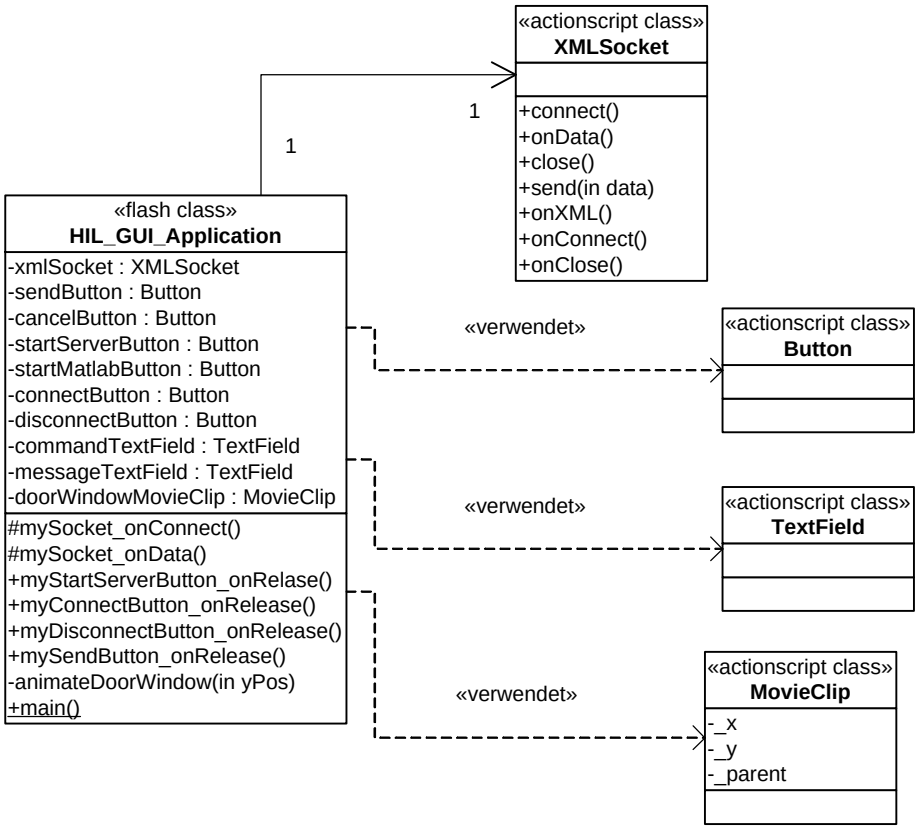

Abb. 5: UML – Klassendiagramm zum Flash-Subsystem

Eine typische Flash-basierte GUI beinhaltet u.a. eine Menü-Struktur, verschiedene Schaltflächen und Eingabefelder zur interaktiven Bedienung, Steuerung und animierten Anzeige der Daten des Simulationsmodells. Derartige Komponenten werden in Flash als Objekte entsprechender ActionScript 2.0-Klassen realisiert (s. Abb. 5). ActionScript 2.0 ist die in Flash integrierte ECMAScript 4 – kompatible Skriptsprache [6].

In Flash wird zur Implementierung des Socket-Clients die XMLSocket-Klasse benutzt (s. Abb. 5). Die XMLSocket-Klasse verwendet dabei zur Kommunikation mit dem Socket-Server XML-Nachrichten, die mit einem Null-Byte ('\0') abgeschlossen sein müssen. Nachdem mittels der Methode connect() eine Socket-Verbindung zum Socket-Server hergestellt wurde, können nun über die beiden Methoden send() bzw. onData() beliebig viele XML-Nachrichten gesendet bzw. empfangen waren.

Die Methode onData() fungiert hierbei als Event-Handler (Ereignisprozedur), die immer dann aufgerufen wird, nachdem eine mit dem Null-Byte abgeschlossene XML-Nachricht über den Socket-Eingabestrom vom TCP/IP-Server empfangen wurde. Durch Überschreiben der Methode onData() lässt sich anwendungsspezifisches Verhalten implementieren, dass durch den Empfang von XML-Nachrichten ausgelöst wird.

Um die im Rahmen einer Matlab-Modellsimulation erzeugten Daten im Flash-Player [7] darstellen und animieren zu können, werden die Simulationsdaten mit Hilfe der Methode onData() ereignisgesteuert empfangen und veiterverarbeitet.

Ein Beispiel soll das prinzipielle Vorgehen weiter verdeutlichen. Für das Folgende werde vorausgesetzt, dass die Simulationsdaten die Bedeutung von Positionsdaten eines Objektes besitzen. Um die Positionsänderungen des Objektes in Flash darzustellen, wird das Objekt geometrisch durch eine Instanz der hierfür besonders geeigneten ActionScript-Klasse MovieClip repräsentiert, die u.a. über die Positionsattribute (_x, _y) verfügt (s. Abb. 5). Ein MovieClip-Objekt lässt sich im Flash-Player jetzt dadurch animieren, indem die (_x, _y) – Werte laufend mit denen von onData() empfangenen Positionsdaten überschrieben werden.

6 Zusammenfassung

Der hier vorgestellte Lösungsansatz zur Kopplung einer Flash-basierten GUI mittels eines zentralen TCP/IP-Socket-Servers an Matlab erlaubt nicht nur die bidirektionale Kommunikation über den Austausch von XML-Nachrichten, sondern ist wegen der Verwendung von permanenten Socket-Verbindungen genauso gut für parallele, verteilte und internet-fähige Simulationsanwendungen geeignet, die darüber hinaus über sehr kleine Latenzzeiten verfügen sollen, also einen echtzeitnahen Betrieb ermöglichen.

Er ist damit aber auch für den Parallelbetrieb von Clientapplikationen wie z. B. mehrerer Matlab-Instanzen selbst auf einem Rechner anwendbar.

In einer typischen HIL – Anwendung ist das Modell die simulierte Arbeitsumgebung eines Steuergerätes, dessen Funktionsweise einem Dauertest bei definierten Bedingungen unterliegen soll.

Die GUI beinhaltet üblicherweise eine Menü-Bedienstruktur zur Parametrierung von Anfangsbedingungen, Buttons und Eingabefelder für die Interaktion während des Modelllaufes sowie die Animation von Bewegungsabläufen des Modells.

Solche Menüstrukturen und Ein-/Ausgabefunktionen können recht problemlos mit dem Matlab-GUI-Editor erstellt werden. Die grafische Darstellung der Elemente der Oberfläche ist dabei jedoch vorgegeben. Außerdem sind Animationen nur eingeschränkt möglich. Hier liegt ein klarer Vorteil für die Anwendung anderer Lösungen, die jedoch eine Schnittstelle zu Matlab voraussetzen.

Bei Designern ist Flash ein verbreitetes Gestaltungsprogramm und besitzt klare Vorzüge, wenn eine professionelle Grafikgestaltung erwünscht ist. Ein Vorteil von Flash für die Anwendung in verteilten Strukturen ist weiterhin die integrierte Netzwerkfunktionalität und XML-Fähigkeit. XML-Strukturen lassen sich über die Socket-Schnittstelle von Flash aber auch über Dateizugriff (im Netzwerk) verarbeiten. Zudem sind auch für Matlab entsprechende Tools für die Konvertierung von XML in Matlab-Strukturen und umgekehrt verfügbar. Von Flash aus können Client-Applikationen, wie auch Matlab, gestartet und beendet werden.

Über die Socket-Verbindungen des Servers kann der Betrieb der HIL - Anordnung über Netzwerk auf verschiedene , räumlich getrennte Komponenten verteilt werden. So kann eine Aufteilung auf Target - Rechner mit dem Modell, Hostrechner mit Matlab und Bedienrechner mit Flash erfolgen.

Unter Einbeziehung von xPC-Target aus der Matlab/Simulink-Toolkette verwaltet ein Hostrechner, auf dem das Simulator_Application - Programm läuft, das Target-Modell, das hier erstellt, modifiziert anschließend in C-Code übersetzt und mit einem Real-Time-Betriebssystem auf einen Target-Rechner geladen werden kann. Der Ladevorgang und die Kommunikation zum Target erfolgen durch das processSimulationModel-Unterprogramm.

Diese Struktur sichert Zykluszeiten der Modellausführung im Mikrosekundenbereich bei gleichzeitiger Entkopplung der Bedien- und Anzeigefunktionen vom Target-Rechner. Matlab kann außerdem seine umfangreichen Signalverarbeitungsfunktionen für ein weiteres Processing auf dem Hostrechner zur Verfügung stellen.

Unter Zuhilfenahme der vorliegenden Socket-Schnittstelle ergänzt Flash die Matlab-Applikation mit seinen spezifischen Stärken der grafischen Präsentation, Interaktion und Animation.

Parallel dazu können an den TCP/IP-Server weitere Clientapplikation angekoppelt werden, die beispielsweise als Monitor des Datenaustausches zwischen diesen Komponenten oder als parallele Interaktionsquelle benutzt werden.

Ein Demonstrationsbeispiel zeigt das beschriebene Zusammenspiel dieser Komponenten.

7 Literatur

[1] Java 2 Platform Standard Edition 5.0 (J2SE 5.0), http://java.sun.com/j2se/1.5.0/download.jsp

[2] Macromedia Flash, http://www.macromedia.com/

[3] Objektorientierung in Flash / ActionScript 2.0, http://www.macromedia.com/devnet/mx/flash/articles/converting_actionscript2.html

[4] Objektorientierung in Matlab, http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/ch11_mat.html#matlab_classes_and_objects

[5] Matlab, Simulink und xPC-Target, http://www.mathworks.de

[6] ECMAScript 4 Netscape Proposal, www.mozilla.org/js/language/es4, http://www.ecma-international.org/publications/standards/Ecma-262.htm

[7] Macromedia Flash Player, http://www.macromedia.com/shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash