Groovy Web Shell

Seminar Social Web Applications Engineering

Markus Kahl
markus.kahl@student.hpi.uni-potsdam.de

Robert Pfeiffer
robert.pfeiffer@student.hpi.uni-potsdam.de

31. Januar 2010

Zusammenfassung

Groovy Web Shell ist eine Webanwendung, die es ihren Benutzern erlaubt, Code auf einem Server auszuführen, zu speichern und von verschiedenen Rechnern aus weiterzuentwickeln. Die Architektur der Groovy Web Shell beruht auf modernen Web-Sandards wie JSON und HTML 5. Dadurch ist es möglich, die Webshell mit verschiedenen voneinander unabhängigen Frontends auszustatten.

Inhaltsverzeichnis

1 Einleitung

Groovy Web Shell ist ein Projekt, welches im Rahmen des Seminars „Social Web Application Engineering“ im Laufe eines Semesters entwickelt wurde. Zu Beginn des Projektes wurden dabei folgende Ziele ins Auge gefasst:

- Ausführung von Groovy-Code in einer Web-Oberfläche
- zentrales Speichern von Code auf dem Server
- persistenter Zustand von Programmen in Sessions
- Semantic-Web-Funktionen
- Visualisierung von Datenstrukturen (Objekten)

Im folgenden wird nun dargelegt, in wie fern diese Ziele erfüllt wurden. Außerdem werden einige der verwendeten Technologien näher erläutert.

Um die Webshell herum ist eine kleine Website entstanden, die auch die Oberfläche zur Web-basierten Ausführungen von Groovy-Code enthält. Man kann sich als Nutzer der Website registrieren und dann Code-Schnipsel veröffentlichen, die widerum von anderen Nutzern kommentiert und auch ausgeführt werden können.

Programme werden innerhalb von Sessions ausgeführt über welche hinweg der Zustand — gebundene Variablen, Felder von Objekten, Funktions- und Klassendefinitionen — erhalten bleiben. Jede Session hat eine einzigartige ID unter deren Angabe eine Session auch wieder aufgenommen werden kann, solange sie nicht abgelaufen ist.

So kann man z.B. etwas programmieren und einem anderen Nutzer die Session schicken, damit er ausgehend vom erzeugten Zustand weiter arbeiten kann. Danach kann man den entstandenen Code nachbearbeiten und als Codeschnipsel veröffentlichen.

Die Groovy Webshell kann außerdem ähnlich wie „Try Ruby“[1] dazu verwendet werden, potentiellen Nutzern schnell und ohne Installation Groovy-Programmierung näher zu bringen.

2 Architektur

Die Architektur der Groovy Web Shell ist grob aufgeteilt in Frontend und Backend. Diese können in voneinander unabhängigen Buildprozessen gebaut und ebenfalls unabhängig voneinander ausgeführt werden.

Der User interagiert im Normalfall sowohl mit dem Grails-Frontend als auch mit dem Backend. Das Grails Frontend stellt eine Benutzerschnittstelle zur Verfügung, die per JSON-Request mit der REST-Schnittstelle des Backends kommuniziert.

Aufgaben des Frontends sind dabei Benutzerverwaltung, Speichern von Code und ausliefern von statischen Inhalten. Aufgaben des Backends sind Sitzungsverwaltung, Ausführen von Code und Rendern von Ergebnissen.

Als „Rendering“ wird im Folgenden die Darstellung als HTML jener Objekte, die von den ausgeführten Skripten zurückgegeben werden, bezeichnet. Das Rendering ist im Backend realisiert, so dass dem Client das gerenderte Ergebnis jedes Skripts zur Verfügung steht.

Die Sitzungsverwaltung umfasst die Speicherung von Code und gerenderten Ergebnissen in der History.

2.1 Vorteile dieser Architektur

Das Frontend ist unabhängig vom Backend. Das bedeutet konkret, dass andere Frontends ohne Veränderungen am Backend benutzt werden können. Als Proof-of-Concept wurde deshalb ein Frontend als Widget und ein „Google Wave“-Robot implementiert.

Das Backend kann mit einem restriktiven SecurityManager versehen werden, um Schäden durch böswillige Benutzer vorzubeugen. Auch die Ausführung des Backends innerhalb eines chroot-Jails oder eines komplett virtualisierten Betriebssystems ist damit möglich.

[1] http://tryruby.org

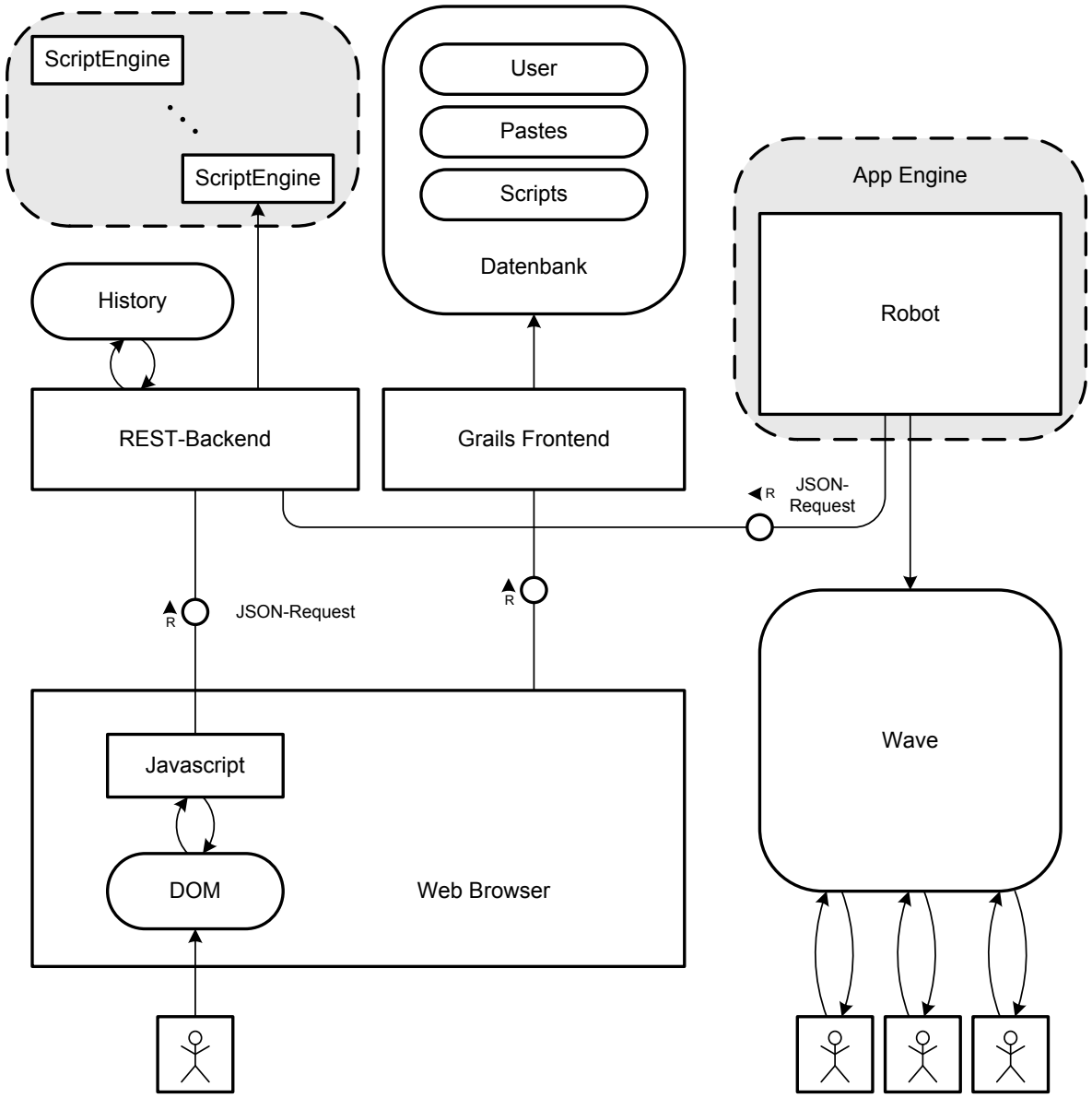

Abbildung 1: Architektur

Das Backend ist nicht an die Programmiersprache Groovy oder das Grails-Framework gebunden. Das Backend lässt sich in jeder Programmiersprache implementieren, die REST-basierte Webservices unterstützt. Damit wäre die Integration eines neu geschriebenen Backends zum Beispiel für Squeak oder F# ohne Änderungen am Frontend denkbar.

In unserer Implementierung wurde Clojure als Programmiersprache für das Backend verwendet. Clojure läuft auf der Java VM und ist vollkommen zu Java kompatibel. Damit ist es möglich, Java-Technologien wie das Java-Scripting Framework und Servlets im Backend zu verwenden.

3 Frontend

Die Website und die integrierte Benutzeroberfläche für die Web Shell sind mit Hilfe von des MVC-Web-Frameworks Grails (ursprünglich Groovy on Rails), HTML, CSS und Javascript implementiert. Die Kommunikation zwischen Web-Oberfläche und Backend der Shell verläuft über AJAX . Dafür haben wir das Javascript-Framework Prototype genutzt.

Gibt der Nutzer Code ein und bestätigt die Eingabe, wird der geschriebene Code zusammen mit der aktuellen Session-ID per AJAX-Request an das Backend geschickt. Daraufhin führt das Backend den Code aus, und antwortet in Form eines JSON-Objektes, in welchem sich noch einmal der ausgeführte Code sowie dessen Ausgabe befindet. Beides wird daraufhin in der Web-Oberfläche angezeigt.

Gespeicherte Scripts können von anderen Nutzern betrachtet und kommentiert werden. Für das Betrachten von Code ist Syntax Highlighting sehr praktisch. Glücklicherweise gibt es dafür eine unter der LGPL 3 stehende Javascript-Bibliothek mit dem treffenden Namen „SyntaxHighlighter“[2] von Alex Gorbatchev.

Grails-Anwendungen werden zu herkömmlichen WAR-Dateien gepackt und können so auf beliebigen Servlet-Containern (wie z.B. Tomcat) ausgeführt werden.

[2] http://alexgorbatchev.com/wiki/SyntaxHighlighter

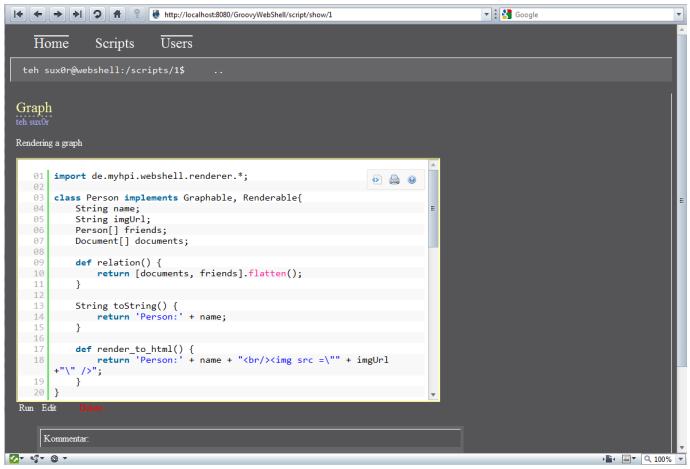

Abbildung 2: Pastebin

4 Backend

Das Backend ist in der Programmiersprache Clojure, einem LISP-Dialekt, implementiert. Genau so wie das Frontend läuft es auf der Java-Virtual Machine und kann in einem Servlet-Container wie Tomcat ausgeführt werden. Features von Clojure wie Protokolle, Software Transactional Memory(STM) und dynamisch gebundene Variablen vereinfachten die Implementierung des Backends sehr. Die Anbindung an die JVM erlaubt trotzdem die Wiederverwendung von bestehendem Code wie dem Java Scripting Framework und den Zugriff auf Groovy-Objekte.

Für die Implementierung des Servlets und der REST-Schnittstelle wurde das Web-Framework Compojure benutzt.

Für die Erzeugung von WAR-Dateien, mit denen das Backend unter einem Servlet-Container ausgeführt werden kann, wurde ein Plugin für das Clojure-Buildsystem „Leiningen“ programmiert, das unter http://clojars.org/lein-war verfügbar ist.

4.1 REST-Schnittstelle

Das Backend wird über eine REST-Basierte Schnittstelle angesprochen. Dabei ist jede Session eine Ressource. Da die PUT-Operation Für Scriptengines keinen Sinn ergibt, werden neue Sitzungen durch ein POST-Request an die URL backend/scripts/ erzeugt. Dabei wird die ID der neuen Sitzung zurückgegeben. Die neue Sitzung befindet sich dann in backend/scripts/*ID*/. Code wird in einer Session durch ein POST-Request an diese URL mit dem Code als Parameter ausgeführt. Ein GET-Request gibt die History der Shell zurück. Die REST-Routen sind im Namespace de.myhpi.webshell.rest festgelegt.

4.2 Rendering

Das Rendering ist nicht im Frontend, sondern im Backend realisiert, da diese Objekte dem Frontend nicht zur Verfügung stehen. Der Client selber kann die Ergebnisse nicht rendern, weil er dazu vielleicht auf einen großen Teil des Objektgraphen und der Klassenhierarchie, der bei der Ausführung des Skriptes entsteht, zugreifen muss, selbst wenn das Ergebnis des Renderings klein ausfällt.

Der Rendering-Algorithmus im Backend muss 3 Anforderungen erfüllen:

1. Aus dem gerenderten Objekt wird ein einzelner String erzeugt, der als JSON-String kodiert übertragen werden kann
2. Der String ist valides HTML, das problemlos in einen DOM-Baum eingefügt werden kann.
3. Das gerenderte Objekt kann ohne weitere AJAX-Anfragen, eingebundene Skripte und generierte Datei-URLs oder sonstige Seiteneffekte dargestellt werden.

4.2.1 Wahl des Rendering-Algorithmus

Wie ein Objekt gerendert wird, hängt von den Interfaces ab, die es implementiert. Ein Interface in Java beschreibt, dass die Instanzen einer Klasse bestimmte Operationen unterstützen, ohne eine Implementierung vorzuschreiben. Damit besitzen Objekte, die ein Interface implementieren, die gleiche Sematik und werden dementsprechend gleich gerendert.

Beispielsweise werden alle Instanzen von java.util.Collection als unnummerierte HTML-Listen dargestellt.

```
1 ["foo","spam",42]
```

Listing 1: Groovy-Array; implementiert java.util.Collection

```
1 <ul>
2     <li>&amp;quot;foo&amp;quot;</li>
3     <li>&amp;quot;spam&amp;quot;</li>
4     <li>42</li>
5 </ul>
```

Listing 2: Gerenderte Liste

4.2.2 Implementierung

Das Rendering ist in den Namespaces de.myhpi.webshell.rendering und de.myhpi.webshell.cont implementiert. Um ein Objekt zu rendern wird die Methode render-to-html des Interfaces de.myhpi.webshell.renderer.Renderable auf diesem Objekt aufgerufen. Mit Hilfe eines Clojure-Protokolls[3] werden Implentierungen dieser Methode zu gängigen Interfaces aus Java hizugefügt.

```
1 (defprotocol Renderable
2   (render-to-html [self]))
3
4 (extend-protocol Renderable
5    java.util.Collection
6    (render-to-html [self]
7      (html (into [:ul]
8           (for [el self] [:li (render-recursive el)])))))
```

Listing 3: Rendering für alle Collecitons

Damit werden diese Interfaces praktisch in Mixins umgewandelt, die eigenes Verhalten zu den implementierenden Klassen hinzufügen.

Diese Rendering-Mixins wurden implementiert:

groovy.lang.GroovyObject Jedes Groovy-Objekt wird als Auflistung aller Namen, Typen und Werte seiner Felder dargestellt. Die Felder werden über die Metaproperties der Metaklasse des Objektes bestimmt. Die Werte der felder werden wiederum gerendert.

groovy.lang.Closure Eine Closure wird als HTML-Formular gerendert. Wenn das Formular abgesendet wird, wird der Code der Closure ausgeführt und der Rückgabewert der Closure ist das Ergebnis des Formulars.

java.util.Collection Eine Collection wird als ungeordnete HTML-Liste ihrer Elemente dargestellt.

java.util.Map Die Darstellung von Maps ist eine HTML-Tabelle ihrer Schlüssel-Wert-Paare.

java.awt.Image Bilder werden als Bilder angezeigt. Dazu wird der Inhalt des Bildes erst in das PNG-Format und anschließend als HTML5-Data-URL kodiert.

[3] Eine genaue Beschreibung von Protokollen findet man unter http://www.assembla.com/wiki/show/clojure/Protocols

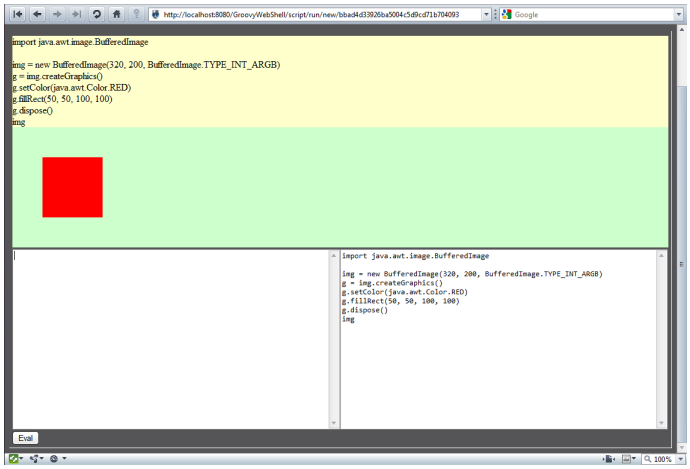

Abbildung 3: Gerendertes Bild

de.myhpi.webshell.renderer.Graphable Das Interface Graphable beschreibt Objekte, die Knoten in einem Graphen sind. Diese Objekte werden als Bild von dem Teil des Graphen gerendert, der von dem Objekt aus erreichbar ist.

java.lang.Object Falls das Objekt keine dieser Interfaces implementiert, wird eine String-Beschreibung mit der toString()-Methode des Objektes erzeugt.

Eine Klasse kann Renderable auch selbst Implementieren:

```
1 import de.myhpi.webshell.renderer.Renderable
2 class Foo implements Renderable {
3    String render_to_html(){
4       return '<b>␣Foo␣</b>';
5    }
6 }
```

Listing 4: explizites Rendering

4.3 Ausführung von Code

Die Ausführung von Code ist im Namespace de.myhpi.webshell.session implementiert.

4.3.1 Java Scripting Framework

Das Java-Scripting Framework beschreibt um eine einheitliche Schnittstelle, mit der man eine Skriptsprache zu Programmen hinzufügen kann. Diese Schnittstelle wird von vielen Programmiersprachen [4], die auf der Java Virtual Machine laufen, implementiert. Die Spezifikation dieser Schnittstelle wurde als JSR 223 [5] im Java Community Process entwickelt. Seit Java 6 ist das Scripting Framework in der Standardversion von Java enthalten.

Das Java Scripting Framework stellt die Klasse javax.script.ScriptEngineManager zur Verfüging, die als Factory für Skriptsprachen agiert. Diese implementieren das Interface javax.script.ScriptEngine und gegebenenfalls auch Compilable oder Invocable. Damit eine ScriptEngine erzeugt werden kann, musse ihre Klasse bereits geladen oder im Classpath und beim Scriptenginemanager registriert sein.

Eine ScriptEngine besitzt eine Methode eval(), die einen Code-String ausführt. Die Semantik dieser Methode eval hängt natürlich von der Skriptsprache ab. Bei Skriptsprachen, die auf der Java-VM laufen, können die Ergebnisse des ausgeführten Codes direkt ohne Marshalling aus dem Skript zurückgegeben werden und Seiteneffekte können die Objekte des aufrufenden Programms verändern.

[4] Auf http://scripting.dev.java.net kann man die Liste von Programmiersprachen, die das Scripting Framework offiziell unterstützt, nachlesen.

[5] http://www.jcp.org/en/jsr/detail?id=223

Es gibt aber auch Scriptengines für Sprachen wie Squeak oder AppleScript, bei denen das nicht der Fall ist.

```
1 import javax.script.*;
2 javascript = new ScriptEngineManager()\
3           .getEngineByName("javascript");
4 println javascript.eval("3␣+␣4");
```

Listing 5: Aufruf von JavaScript aus Groovy

Prinzipiell könnte jede Programmiersprache, die mit dem Scripting Framework funktioniert, in der Webshell eingesetzt werden, indem der Parameter *language* im session-namespace umgestellt wird. Dazu muss diese Sprache jedoch auch ihren Zustand in der ScriptEngine speichern, anstatt global oder gar nicht, und sollte sinnvolle Java-Objekte zurückgeben. Verschiedene Programmiersprachen könnten außerdem andere Operationen benötigen, die der Securitymanager blockiert, oder nur auf bestimmten Plattformen verfügbar sein, wie beispielsweise AppleScript.

Deswegen sind für die Verwendung anderer Sprachen als Groovy vielleicht Anpassungen erforderlich. Wir haben testweise JavaScript und JRuby in der Webshell ausgeführt.

4.3.2 Clojure STM

Für jede Sitzung wird eine zustandsbehaftete Scriptengine und eine Liste von Ein- und Ausgaben(History) mit einer zufälligen ID erzeugt und in einer Map gespeichert. Neue Sitzungen werden Thread-sicher innerhalb von Transaktionen erzeugt. Jede Sitzung ist innerhalb eines Atoms gespeichert. Atome sind Objekte, die eine Referenz auf einen Wert haben, die nur synchron und atomar verändert werden kann. Damit ist ausgeschlossen, dass zwei Skripte gleichzeitig in einer Scriptengine ausgehührt werden, oder dass es zu Race Conditions beim Speichern der History kommt.

4.3.3 Java Security Manager

Es dem Nutzer zu ermöglichen, beliebige Scripts auf dem Server auszuführen birgt Risiken. So könnte man z.B. auf das Dateisystem des Servers zugreifen, um dort Dateien auszulesen oder gar um sie zu löschen.

Das muss verhindert werden. Hier kommt der Java Security Manager ins Spiel. Mit Hilfe des Java Security Managers können Rechte für Code festgelegt werden. Wird nun Code ausgeführt, der eine nicht gestattete Aktion versucht, wie z.B. das Lesen der Festplatte, so wird eine SecurityException geworfen und der Zugriff somit verhindert.

Um Rechte festzulegen, könnte man seine eigene Policy-Klasse schreiben, welche von java.security.Policy erben muss, und diese bei Programmstart einsetzen. Ebenso könnte man direkt seinen eigenen SecurityManager schreiben. Davon haben wir allerdings abgesehen. Eine andere Möglichkeit ist es nämlich, die sogenannten Policy-Dateien zu nutzen. In diesen Text-Dateien werden „Permissions“ verteilt. Beispiele hierfür sind die folgenden:

AllPermission Zugriff auf alles

FilePermission Zugriff auf Dateien und Verzeichnisse

NetPermission Zugriff auf Netzwerkressourcen

PropertyPermission Zugriff auf Systemeigenschaften

ReflectPermission Zugriff über Reflection auf andere Objekte erlauben

RuntimePermission Einschränkungen von Laufzeitsystemen wie ClassLoader

SerializablePermission Beschränkung der Serialisierung

SocketPermission Spezielle Einschränkungen an einer Socket-Verbindung

Diese Berechtigungen können für bestimmten Code in grant-Anweisungen innerhalb der Policy-Dateien vergeben werden. Dabei gibt es drei Möglichkeiten, diesen Code zu identifizieren:

1. Codebase
2. Signierung
3. Principal

Über die Codebase wird Code, der von einem bestimmten URL stammt, Rechte gewährt. Außerdem kann man Rechte für signierten Code vergeben. Signierter Code stammt aus signierten JARs. Die Tools zum Signieren von Jars liegen dem JDK bei. Principal ermöglicht eine komplexe Rechtevergabe auf Rollenbasis unter Nutzung des JAAS (Java Authentication and Authorization Service). Darauf wollen wir jedoch nicht weiter eingehen, da man zu diesem Thema alleine eine eigene Ausarbeitung schreiben könnte.

Die einfachste Möglichkeit aus Entwicklersicht ist der Weg über die Codebase. Eine entsprechende grant-Anweisung sähe wie folgt aus:

```
1 // GroovyWebShell
2 grant codeBase
      "file:${catalina.home}/webapps/GroovyWebShell/-" {
3     permission java.security.AllPermission;
4 };
```

Listing 6: Konfiguration des SecurityManagers

Dies ist ein Ausschnitt aus der Policy-Datei für unsere Groovy Web Shell. Bei dieser Anweisung wird in Zeile 2 mittels "codeBase 'PFAD'" festgelegt, dass die folgenden Berechtigungen nur für Code mit der entsprechenden Codebase gelten.

Ist der Security Manager aktiviert, wird standardmäßig jede sicherheitskritische Aktion verboten. D.h. man muss alle Aktionen, die benötigt werden, explizit gestatten.

In unserem Fall verleihen wir allem Code unser eigenen Anwendung einfach sämtliche Berechtigungen. Der von den Nutzern ausgeführte Code hat eine andere Codebase, da es sich dabei immer um zur Laufzeit compilierte Groovy-Skripte handelt. Diese haben automatisch die Codebase "file:/groovy/script".

Da standardmäßig nichts erlaubt ist, müssen wir zuerst einmal Groovys „Grundrechte“ sicherstellen:

```
1 // groovy scripts
2 grant codeBase "file:/groovy/script" {
3     permission java.lang.RuntimePermission
          "defineClassInPackage.java.io";
4     permission java.lang.RuntimePermission
          "defineClassInPackage.java.lang";
5     permission java.lang.RuntimePermission
          "defineClassInPackage.java.net";
6     permission java.lang.RuntimePermission
          "defineClassInPackage.java.util";
7 };
```

Listing 7: Rechte für Groovy

Die Berechtiungen in Zeile 3 bis 6 sind nötig, damit der Nutzer überhaupt in Groovy programmieren kann, denn Groovy benutzt und erweitert die Standard-Java-Klassen. Würden wir diese Berechtigungen nicht gewähren, könnte man als Nutzer z.B. nicht einfach Code wie den folgenden schreiben:

```
1 String text = new File("text.txt").getText()
```

Listing 8: Einlesen einer Datei

Groovy erweitert die Klasse File um die sehr hilfreiche Methode #getText(), um dem Programmierer das Leben leichter zu machen. Ohne diese Erweiterung müsste man dasselbe auf die folgende Weise bewerkstelligen,

```
1 StringBuilder sb = new StringBuilder()
2 BufferedReader in = null;
3 try {
4    in = new BufferedReader(new FileReader("text.txt"));
5    String line;
6    while ((line = in.readLine()) != null) {
7       sb.append(line);
8       sb.append(System.getProperty("line.separator"));
9    }
10 } catch (IOException e) {
11    e.printStackTrace();
12 } finally {
13    if (in != null) {
14       try {
15          in.close();
16       } catch (IOException e) { }
17    }
18 }
19 String text = sb.toString();
```

Listing 9: Einlesen einer Datei ohne Groovy-Methoden

Was leider nicht ganz so kompakt ist[6] und womit die geschrieben Skripte erheblich länger werden würden.

Was man mit dem Java Security Manager nicht tun kann, ist explizit Rechte zu verweigern. So kann man z.B. nicht den Zugriff auf bestimmte Pakete steuern. Nur die sun-internen Pakete wie sun.java2d und dergleichen werden durch den Security Manager geschützt und sind standardmäßig nicht verfügbar.

Wenn man also AWT-Klassen verwendet, die intern auf solche Pakete zugreifen (wie z.B. java.awt.Graphics), wird dies an einer Exception wie der folgenden scheitern:

```
1 java.security.AccessControlException: access denied
      (java.lang.RuntimePermission
      accessClassInPackage.sun.java2d)
```

Listing 10: Fehlermeldung

Alle anderen Pakete (java.**, javax.**, usw.) sind ohne Zugriffsprüfung verfügbar. Das ist z.B. bei den Paketen AWT und Swing ein Problem, wenn die JVM auf einem Server mit einem Display läuft. Denn dann kann der Nutzer einfach munter Fenster auf der Serverseite erstellen. Es wird nur der Zugriff auf das System-Clipboard, die Pixel der Anzeige und System Events beschränkt.

Um alle anderen Probleme müssten wir uns also selbst kümmern. Eine Möglichkeit ist es einfach den vom Nutzer eingegebenen Code nach imports oder expliziten Nutzungen von Klassen aus unerwünschten Paketen zu durchsuchen und entsprechende Stellen herauszufiltern.

Wenn das Backend allerdings auf einem Server läuft, der keine Anzeige hat, schlagen solche Versuche mit dem AWT und Swing jedoch sowieso automatisch fehl und da das Backend auch tatsächlich auch auf einem Server ohne Display laufen soll, müssen wir uns darum nicht weiter kümmern.

5 Alternative Frontends

5.1 Google Wave

Google Wave(http://wave.google.com) ist eine Browser-Basierte Kollaborationstechnologie, die von Google entwickelt wurde und Features von Wikis, Email und Chat in sich vereint. Dokumente werden wie bei Wikis zentral gehostet und kollaborativ bearbeitet, aber Änderungen sind für alle Teilnehmer in Echtzeit sichtbar. Ein Dokument, genannt „Wave“, ist in einer Baumstruktur aufgebaut. Es kann mehrere „Wavelets“ mit verschiedenen Teilnehmern enthalten. Einzelne Absätze werden „Blip“ genannt. Um zusätzliche Features zu einer Wave hinzuzufügen, kann man Robots[7] als Teilnehmer einer Wave hinzufügen. Robots werden per JSON-RPC über Events in der Wave informiert und können auf die gleiche Weise Waves lesen und schreiben. Google stellt für die Entwicklung von Robots ein einfaches Java-basiertes objektorientierts API bereit. http://code.google.com/apis/wave/extensions/robots/

[6] Mit Java 1.5 und der neuen Klasse java.util.Scanner geht das mittlerweile auch etwas kürzer; man muss jedoch nach wie vor Zeile für Zeile selbst auslesen.

[7] http://code.google.com/apis/wave/extensions/robots/

Der Google Wave Robot in examples/wavebot.clj wartet auf Events und beantwortet jeden geschriebenen Blip, der den String „evalgroovy“ enthält, indem es den Code darin ausführt und einen neuen Blip aus dem Ergebnis erstellt.

Aus Sicherheitsgründen können Robots für Google Wave bisher nur auf der Google App Engine ausgeführt werden. Der Robot läuft daher momentan auf rfgpfeiffer.appspot.com.

Die Google App Engine ist eine Cloud Computing-Plattform, die selbst starken Sicherheitsbeschränkungen unterliegt, daher ist die Groovy Web Shell dort nicht ohne Anpassung lauffähig. So können zum Beispiel keine Threads erzeugt und kein STM verwendet werden. Aller Java-Code muss vorher zu Bytecode kompiliert worden sein. Die Anwendungen werden im WAR-Format auf die App Engine hochgeladen und können nur als Servlets ausgeführt werden.

Daher ist die Version auf rfgpfeiffer.appspot.com an die Google App Engine angepasst.

5.2 Widget

Eine Einbindung der Shell in eine HTML-Seite ist beispielhaft im examples/embed.html implementiert. Das Backend wird direkt per Ajax angesprochen; dabei wird die Same Origin Policy des Browsers umgangen, indem der Code vom Backend aus geladen wird.

6 Deployment auf Tomcat

Prinzipiell können sowohl Frontend als auch Backend auf einem beliebigen aktuellen Servlet Container (Tomcat, Jetty, GlassFish, Weblogic, ...) eingesetzt werden. Beim Frontend gibt es diesbezüglich auch nichts weiter zu beachten. Je nach Servlet Container muss man die GroovyWebShell.war und die webshell-backend.war einfach in das autodeploy-Verzeichnis kopieren oder kann die Anwendungen auch über eine Web-Oberfläche starten.

Eine Hürde besteht jedoch noch beim Backend: die Sicherheit. Da wir das Backend mit aktiviertem Java Security Manager laufen lassen wollen, müssen wir sehen, wie wir ihn auf dem Ziel-Servlet-Contaiener aktivieren.

Letztendlich soll das Backend auf einem Tomcat laufen. Um nun dort seine Policies hinzuzufügen, kann man sie einfach in die Datei catalina.policy, welche sich im conf-Verzeichnis der Tomcat-Installation befindet, eintragen.

Tomcat wird standardmäßig ohne aktivierten Security Manager gestartet. Also muss er, damit die Sicherheitseinstellungen auch Anwendung finden, im „sicheren Modus“ neugestartet werden.

Dazu reicht es aus, beim Start des Servers einfach die Option -security anzugeben:

```
1 catalina start -security
```

Anschließend können Sie mit ihrem Browser die URL http://*Servername*/GroovyWebShell öffnen.
Happy Hacking!