WebGIS

Einleitung

Das OpenInfRA-WebGIS wird als Plugin/Erweiterung zum Basissystem entwickelt. Die Installation soll zukünftig entsprechend der Vorgaben es Basissystems erfolgen (Definition Plugin ausstehend).
Das WebGIS ist in erster Linie als eine JavaScript-Anwendung konzipiert. D.h., dass der überwiegende Teil der Logik auf Seiten des Clients im Sinne einer RIA implementiert ist. Konsumiert werden in erster Linie standardisierte OWS-Schnittstellen wie WMS und WFS. So ist es möglich den Client auf beliebige Geodatenservices zuzugreifen.

Konzeptuelle Überlegungen

Mapping OGC-Layer/OpenInfRA-Instanzen

- der Zugriff seitens des WebGIS-Clients und externe GIS-Tools wird über OGC-konforme Endpunkte erfolgen (WMS, WFS)
- Themenausprägungen werden dabei als OGC-Layer interpretiert. (Bsp. Themenausprägung "Scherbe" === WMS Layer "Scherbe")
- Themeninstanzen stellen Feature der Layer dar
- Nachfolgende Entwicklungen sollen die Definition mehrerer Layer zu einer Themenausprägung anhand von Attributen (Filter) der Themenausprägung erlauben (Bsp. Themensausprägung "Scherbe" umfasst Layer "Scherbe", "Scherbe - Römerzeit"; 1:n-Zuordnung zu Themenausprägung)
- Nachfolgend kann die Bildung von übergreifenden Gruppen über Themenausprägungen umgesetzt werden (WMS-Gruppenlayer "Römerzeit" umfassend "Scherbe - Römerzeit", "Münze - Römerzeit")
- Schnittstellen zu OpenInfRA
    - Für die Verwaltung der GeoServer Layer ist das globale Deaktivieren von OGC-Services hinderlich, folglich werden alle Dienste des OWS aktiviert sein
    - Das Freigeben von und Deaktivieren von Layern und Diensten erfolgt impliziet durch das anlegen/löschen von Diensten und die Freigabe von Endpunkten über GeoServer-Nutzerrechte (Public, Projektbearbeiter, ...)

Mapping Themenausprägung - Themeninstanzen

xxx === Themenausprägung auf die sich die Georeferenzierung bezieht

- Themenausprägung
  - de:Georeferenzierung_xxx, eng:georeferencing_xxx
- Attributtypgruppe
  - name = de:Georeferenzierung, eng:georeferencing
  - description = de:"Georeferenzierung einer Themeninstanz", eng:"georeferencing of a topic instance"
- Attributtypen (Multiplizität je 0..1 in attribute_type_to_attribute_type_group bzw. 1 für Geometrie)
  - de:gemessen_am, eng:measured_on => date
  - de:gemessen_von, eng:measured_by => text
  - de:Messmethode, eng:method_of_measurement => text
  - de:Geometrie, eng:geometry => geometry(geometry)
- Beziehungstyp
  - de:georeferenziert, eng:georeferenced
  - Multiplizität => 0..1
- Beziehungstyp zu Themenausprägung
  - je der Typ "georeferenziert" zwischen Themenausprägung Georeferenzierung_xxx und Themenausprägung xxx

Eingliederung OpenInfRA-Kernsystem

- Zusätzliche Informationen wie Basislayer etc. werden als eigene Themenausprägungen/Instanzen abgebildet
- das WebGIS wird als "OpenInfra-Extension" implementiert, die Nutzeroberfläche wird entsprechend den Vorgaben des Kernsystems in das Gerüst der Anwendung integriert
- die Administration des Clients erfolgt als Unterseite des durch den AN umzusetzenden Adminclients, die entsprechenden Optionen werden in eine eigene "Unterseite" der Adminoberfläche organisiert.
- der Nutzerzugang auf diese Bereich ist entsprechend den Vorgaben des Kernsystems geregelt

Administration

- die Konfiguration des WebGIS-Clients erfolgt als "Unterseite" des OpenInfRA-Adminclients
    - Auswahl von initialen Basiskarten, Themen (aus zuvor definierten Layern/Themen)
    - Auswahl der zur Verfügung stehenden Steuerelemente (rollenspzifische Ausprägung)
    - initiale Kartenposition
    - etc. (s. AF/Tickets)
- die Konfiguration des OWS (GeoServer) wird in erster Iteration über das GeoServer-Frontend erfolgen bis die nötigen Funktionen zum Anlegen von Layern (SQL-Views) etc. ebenfalls im Adminclient abgehandelt werden (Ziel: Projektende)
- Zur Konfiguration des GeoServers über den Adminclient wird dann die Java GeoServer Rest API genutzt

Import

- Shapefile-Import wird durch AN umgesetzt
- Priorisierung bei Umsetzung von Geo-Import
    1. Geodaten über Attribut "Instanz-ID" zu Themeninstanzen zuordnen
    2. Zuordung der Geodaten über Nutzerinteraktion
- Geodaten im Textformat werden über temporäre Layer des Clients zugeordnet
- Perspektivisch ist die serverseitige Konvertierung von Binärdaten (Shapefiles) in Textdateien (GeoJSON) umsetzbar, um ebenfalls über temporäre Layer zu importieren
- Rasterdaten werden über Fileupload geladen (s.a. Anforderung an AN)

Technische Rahmenbedingungen

Im Folgenden sind die aktuell genutzten Komponenten und ihre Lizenzen gelistet.

Kernkomponenten

Komponenten die grundlegender Bestandteil der Anwendung sind und entsprechend auch in Zukunft nicht ohne größeren Aufwand ausgetauscht werden können.

Client

- ExtJS 4.2.1 (GPL3)
  - Dokumentation
  - Code-Mirror
- DeftJS
- GeoExt 2.0.3 (BSD)
- OpenLayers 2.13.1 (BSD)
  - API-Dokumentation
  - User-Dokumentation
  - Repository
  - inkl. Proj4js 1.3 (BSD) (Projektion von Vektordaten)

Server

- Java 7 (Geoserver läuft am besten Sun/Oracle)
- Performance: Sun JAI und JAI Image I/O (Installation)
- GeoServer >2.5.1 (besser >2.7.0)
  - Dokumentation
- OpenInfRA-Basisanwendung

Zusätzliche Komponenten

- ESRI resource-proxy (Apache License 2)
  - Klassische WMS/WFS-Dienste sind oft nicht mit den nötigen CORS-Settings konfiguriert um die Nutzung des Dienstes innerhalb einer Webanwendung außerhalb der Server-Domäne zu ermöglichen. Hierfür kein ein Proxy eingesetzt werden, der AJAX Anfragen etwa für GetCapabilities-Requests über die eigene Domäne umleitet. Im Allgemeinen wird dies ein Konfigurationsproblem bei der Installation der OpenInfRA-Anwendung darstellen. Die Funktionalitäten der Web-Anwendungen sind hiervon nicht direkt betroffen.
- FontAwesome (MIT/SIL OFL)
  - Angestrebt ist die Nutzung eines einheitlichen Icon-Sets über die gesamte OpenInfRA-Anwendung hinweg. Favorisiert ist FontAwesome als CSS-Font. Leider bietet dieses Fontset nur eingeschränkte unterstützung für GIS-relevante Icons, so dass hier auf andere Quellen zugegriffen werden muss.
- Sencha CMD (properität)
  - Sencha CMD ist nicht im Code-Repo hinterlegt und dient ausschließlich zur Entwicklung. Das Hosten der Anwendung ist auch auf Grund der Lizenzbedingungen nicht möglich. Die reine Nutzung verstößt nicht gegen die angestrebte GPL-Lizenz von OpenInfRA, vielmehr bindet die Nutzung des ExtJS Frameworks unter der GPL3-Lizenz den generierten Code an eben diese Ausprägung der GPL.
  - Das Tool dient dem Build minifizierter, konkatinierter und uglifizierter JavaScript- & CSS-Dateien.

Installation

Konfiguration Ubuntu

- (Proxy-Einstellungen beachten)

Installation Oracle/Sun Java 7

- Inoffizielles apt-Repo hinzufügen
- Update & Installation der JRE
- Defaults setzen

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java7-installer
sudo apt-get install oracle-java7-set-default
```

Konfiguration des Tomcat ("Container Consideration")

-server	Enables the server Java Virtual Machine (JVM), which compiles bytecode much earlier and with stronger optimizations. Startup and initial calls will be slower due to “just-in-time” (JIT) compilation taking longer, but subsequent calls will be faster.
-Xmx256M -Xms48m	Allocates extra memory to your server. By default, JVM will use only 64MB of heap. If you’re serving just vector data, you’ll be streaming, so having more memory won’t increase performance. If you’re serving coverages, however, JAI will use a disk cache.
-Xmx256M	allocates 256MB of memory to GeoServer (use more if you have excess memory). It is also a good idea to configure the JAI tile cache size (see the Server Config

	page in the Web Administration Interfacesection) so that it uses 75% of the heap (0.75).
-Xms48m	will tell the virtual machine to grab a 48MB heap on startup, which will make heap management more stable during heavy load serving.
-XX:SoftRefLRUPolicyMSPerMB=36000	Increases the lifetime of "soft references" in GeoServer. GeoServer uses soft references to cache datastore references and other similar requests. Making them live longer will increase the effectiveness of the cache.
-XX:MaxPermSize=128m	Increases the maximum size of permanent generation (or "permgen") allocated to GeoServer to 128MB. Permgen is the heap portion where the class bytecode is stored. GeoServer uses lots of classes, and it may exhaust that space quickly, leading to out of memory errors. This is especially important if you're deploying GeoServer along with other applications in the same container, or if you need to deploy multiple GeoServer instances inside the same container.
-XX:+UseParallelGC	Enables the throughput garbage collector.

Laden von GeoServer als WAR in Tomcat-Container

Hierfür muss GeoServer lediglich als WAR-Archiv ins Verzeichnis webapps der Tomcat-Installation verschoben werden. Der Servlet-Pfad /geoserver wird standardmäßig genutzt.
Zur besseren Migrierbarkeit zwischen verschiedenen GeoServer-Versionen und zur Installation der initialen Geodaten bietet es sich weiterhin an, das eigentliche Datenverzeichnis die durch den GeoServer genutzt wird, seperat festzulegen (Standard: Tomcat-Installation/webapps /geoserver/data). Das nötige vorgehen ist hier beschrieben. (Für eine endgültige Distribution der OpenInfRA-Anwendung erscheint das setzen via Servlet-Context am sinnvollsten, da so sichergestellt werden kann, dass das mitgelieferte Data-Directory eingebunden ist.)

Installation von JAI und JAI ImageIO

Zur Performancesteigerung beim Rendern von Rasterbildern wird die Nutzung der JAI Bibliothek angeraten. Leider ist die Distribution schlecht dokumentiert:

- Download JAI und JAI-ImageIO entsprechend Betriebssystem
- Installation von JAI entsprechend Anleitung
- Installation von JAO-ImageIO entsprechend Anleitung
  - Bei Checksummenfehler bei der Installation von ImageIO:
  - cp jai_imageio-1_1-lib-linux-amd64-jre.bin jai_imageio-1_1-lib-linux-amd64-jre-fixed.bin
  - sed s/+215/-n+215/ -i jai_imageio-1_1-lib-linux-amd64-jre-fixed.bin
  - *-fixed.bin nutzen

Implementation

Die Anwendung folgt strikt den Vorgaben die durch das ExtJS Framework gestellt werden und implementiert dabei die klare Trennung zwischen Darstellung und Datenverarbeitung nach dem Model/View/Controller-Pattern (MVC). Die allgemeine Herrangehensweise ist für ExtJS hier beschrieben.
Da aber die Trennung von Anwendungslogik und einfacher Darstellung durch das Einbinden von OpenLayers-Klassen nicht immer klar möglich ist, wurde auf den Einsatz der ExtJS-Erweiterung DeftJS zurückgegriffen. Diese relativ kleine Bibliothek stellt einen IoC-Container für ExtJS-Anwendungen zur Verfügung der das klassische ExtJS-Application-Objekt ersetzt.

Architektur

Die JavaScript-Anwendung folgt der durch das ExtJS-Framework vorgegebenen Struktur und unterteilt GUI-Komponenten als Views, Datencontainer als Models und implementiert entsprechende Vermittler zwischen diesen Bestandteilen als Controller.

Um die Konfigurierbarkeit zu verbessern und Aufgabenbereiche zwischen den Komponenten abzugrenzen wurde außerdem das Inversion of Control (s.a. Dependency Injection)-Pattern umgesetzt. Hierfür kommt die erwähnte Erweiterung DeftJS zum Einsatz.
DeftJS erlaubt es, ausgehend von einem zentralen Objekt - der Application - sogenannte Service Provider zu definieren, welche zur Laufzeit für andere Objekte zur Verfügung stehen. So ist es im Falle der WebGIS-Anwendung beispielsweise leicht zu gewährleisten, dass es zu jedem Zeitpunkt ausschließlich einen "LayerStore" gibt, welcher über die aktuell zur Darstellung gewählten GIS-Layer verfügt. Andere Komponenten können diesen durch den Anwendungskontext bereitgestellten Service nutzen, indem sie eben auf diesen Provider referenzieren:

```
Ext.define('GIS.view.button.FeatureInfo', {
    extend: 'GIS.view.button.OlButton',

    alias: 'widget.gis_view_button_featureinfo',

    inject: [              // Der Service Provider des Typs "MapService" wird in die Komponente FeatureInfo "inje
        'mapService'       // Zur Laufzeit kann die Komponente via this.getMapService() auf den Provider zugreife
    ],
```

Allgemein Projektstruktur

Die Struktur des Repositories richtet sich im allgemeinen nach den Vorgaben der beteiligten Komponenten.

- build (nur lokale)
  - Enthält den zuletzt ausgeführten Build der JavaScript-Anwendung
- docs
  - API Dokumentation im HTML-Format
- ext
  - Root-Verzeichnis des ExtJS-Frameworks
- GIS

    - WebGIS-Komponenten und Anwendungslogik
- packages
    - ExtJS-Packages die über die Anwendung referenziert werden.

Projektstruktur GIS-Anwendung (WIP)

- Application.js
- app
    - config
    - container
    - container
    - context
    - controller
    - model
    - plugin
    - proxy
    - service
    - store
    - util
    - ux
    - view
        - button
        - buttongroup
        - component
        - container
        - form
        - grid
        - menu
        - panel
        - table
        - toolbar
        - tree
        - Viewport.js
    - Application.js

Build Clientcode

Der javaScript-Code kann über das oben erwähnte Sencha Cmd Tool zu einer fertigen Anwendung kompiliert werden. Dabei muss sich das Tool im Pfad des Nutzers befinden.

Zu Beginn ist außerdem die Erweiterung Deft.js ins Repo der Sencha CMD-Installation zu übernehmen:

```
sencha repo add -address http://packages.deftjs.org/ -name deftjs
```

Danach den Build starten mit:

```
sencha app build
```

Durch den Befehl werden sämtliche durch die Anwendung referenzierten JavaScript-Dateien des ExtJS-Frameworks, der GeoExt-Erweiterung und der eigentlichen Anwendung im webapp/app-Verzeichnis zu einer JavaScript-Datei verschmolzen. Der Befehl ist aus dem Verzeichnis ROOT/GIS auszuführen.

Soll nur der modifizierte SASS-Code zu einer zentralen CSS-Datei gepackt werden, so genügt das Kommando:

```
sencha ant sass
```

Weitere Informationen können in der Dokumentation zu Sencha CMD nachgelesen werden. (Zusammenfassung) (s.a. sencha app watch um einen lokalen Entwicklungsserver zu starten und Änderungen "on-the-fly" zu kom

Build OpenInfRA-Plugin

Im Moment nicht relevant, da entsprechende Schnittstellen zum Kernsystem noch geschaffen werden müssen.

Benutzeroberfläche

Lokalisierung

ein Mechanismus zur Lokalisierung einer ExtJS-Anwendung ist bereits Standardumfang des Frameworks und wird wie in der offiziellen Dokumentation beschrieben zum Einsatz kommen. DIe Grundlegende herangehensweise ist, zu übersetztenden Text als Properties der GUI-Komponenten-Klassen zu hinterlegen und diese dynamisch während der instanziierung zu laden. So ist es möglich eben diese Properties durch Overrides im Paket locale zu überschreiben und auf die aktuelle Anfrage entsprechend zu reagieren.

Nutzung des FontAwesome Iconsets mit ExtJS

Allgemein können Glyphen, wie durch FontAwesome definiert, leicht in ExtJS-Komponenten integriert werden. In der Regel dient hier das Config-Property "glyph". Für einen Button könnte dies beispielsweise so aussehen:

```
Ext.widget('button', {
     text: 'Save',
    glyph: 'xf0c7@FontAwesome' // Hex id des Icons im Iconset
});
```

Um die Nutzung der Icons zu erleichtern und sie auch in Zukunft schnell gegen ein neues Iconset tauschen zu können, bietet sich die Definition an zentraler Stelle an. Hierfür kann leicht eine ExtJS-Singleton-Klasse implementiert werden, die konstante Icon-Werte zur Laufzeit bereitstellt. Bsp.:

```
Ext.define('GIS.util.Glyphs', {
    singleton: true,

    ADD: 0xf067,
    ZOOM_EXTENT: 0xf0b2,
    ZOOM_IN: 0xf00e,
    ZOOM_OUT: 0xf010,

    constructor: function() {
        // Do this so we don't have to add '@FontAwesome' with every glyph
        // definition and can stick to hex ids.
        Ext.setGlyphFontFamily('FontAwesome');
    }
});
```

Um meherer Iconsets gegeneinander testen zu können, bzw. schnell für die gesamte Anwendung zu tauschen, sollte weiterhin in Implementierungsklasse und Interface unterschieden werden. Um unnötige Vererbung zwischen den Klassen zu vermeiden, genügt es einen alternativen, wohlbekannten Klassennamen in den Implemntierungsklassen zu definieren, der schließlich zur Laufzeit durch andere Komponenten für den Zugriff genutzt wird. Bsp:

```
Ext.define('GIS.util.FontAwesomeGlyphs', {

    /**
     * Alternative class names that will be used to call for any glyphs in inline
     * application code. By sticking to GIS.util.Glyphs instead of the implementation
     * class, implementation of this class and the used Fontset is interchangable.
     * Only one implementation should be loaded per application, otherwise definitions
     * will collide.
     * @type {Array}
     */
    alternateClassName: ['GIS.util.Glyphs'],

    singleton: true,

    ADD: 0xf067,
    ZOOM_EXTENT: 0xf0b2,
    ZOOM_IN: 0xf00e,
    ZOOM_OUT: 0xf010,

    constructor: function() {
        // Do this so we don't have to add '@FontAwesome' with every glyph
        // definition and can stick to hex ids.
        Ext.setGlyphFontFamily('FontAwesome');
    }
});
```

Beim Start der Anwendung wird nun die konkrete Implementierung einmalig an zentraler Stelle vorgeladen:

```
Ext.define('GIS.Application', {
    extend: 'Ext.app.Application',
    requires: [
        'GIS.view.Viewport',
        // call implementation class but use GIS.util.Glyphs inline
        'GIS.util.FontAwesomeGlyphs'
    ],
...
```

Und schlielich in der Anwendung genutzt:

```
...
{
            xtype: 'button',
            glyph: GIS.util.Glyphs.EYE,
            tooltip: 'Show legend',
            itemId: 'showLegend',
            cls: 'independentAction'
},
...
```