Softwarestudie

Easy Inference Plugin

Allgemeine Aufgaben

Was ist de Unterschied zwischen den Erweiterungstypen?

Komponenten
Komponenten sind ein MVC-Block, d.h. es gibt die Hauptdatei der Komponente (Controller) und einige Templates die angezeigt werden. Die Komponente wird über URLs im Ontowiki angesteuert und normalerweise im Hauptbereich angezeigt bzw. für asynchrone Anfragen verwendet. Wichtige Arten von Komponenten sind z.B. neue Tabs im Ontowiki oder neue Webservice Endpoints.

Module
Module fügen der Ontowiki Oberfläche zusätzliche Fenster hinzu womit der Benutzer den Inhalt des Hauptfensters beeinflussen kann.

Plugins
Plugins reagieren auf Events und führen bei den Events entsprechenden Code aus. Sie sind also immer dann einzusetzen, wenn man etwas durch die Reaktion auf Events erreichen möchte.

Wie greife ich auf die aktuelle Wissensbasis zu?

Wissensbasen werden grundlegend vom System des Ontowiki selbst geladen und verarbeitet. Somit ist es recht einfach an die aktuelle Wissensbasis zu gelangen. Im Ontowiki selbst gibt es eine Singleton Klasse „Application“. Über die öffentliche Klasse `getInstance()` holt man sich eine aktuelle Instanz der Ontowiki-Anwendungs-Klasse, welche einige Methoden mit sich bringt. Über eine `__get()` Methode kann man dann auf die aktuelle Wissensbasis zugreifen.

```
$_owApp = OntoWiki_Application::getInstance();
$model = $_owApp->__get('selectedModel');
```

Wie greife ich auf die aktuelle Ressource zu?

Parallel dazu kann man ebenfalls auf die aktuelle Ressource zugreifen.

```
$_owApp = OntoWiki_Application::getInstance();
$resource= $_owApp->__get('selectedResource');
```

Wie stelle ich eine SPARQL Anfrage?

Um eine SPARQL Anfrage an das Ontowiki-System zu stellen, kommt man um die „Erfurt“-Engine nicht herum. Diese Engine bietet eine reihe nützlicher Klassen und Methoden, um SPARQL-Anfragen auf die aktuell ausgewählte Wissensbasis zu ermöglichen. Eine einfache Anfrage lässt sich beispielhaft wie folgt realisieren.

```
$query = new Erfurt_Sparql_SimpleQuery();
$query->setProloguePart('SELECT ?objekt ')
      ->setWherePart('WHERE { <subjekt> <praedikat> ?objekt } ');
```

Wie lese ich die private Konfiguration der Erweiterung aus?

Die private Konfiguration lässt sich sehr einfach über folgenden Art und Weise abfragen:

```
$this->_privateConfig;
```

Dies kann schlüsselweise oder über das Zend_Config Objekt abgefragt werden, da dies die toArray-Methode unterstützt. Somit kann man die komplette Konfiguration auslesen und weitervearbeiten.

```
$this->_privateConfig->singleKey; // Schlüsselweise
```

Welche Möglichkeiten gibt es, um Statements zu einer Wissensbasis hinzuzufügen

Das Hinzufügen von Statements zu einer Wissensbasis erfolgt mit Hilfe der `Erfurt_Store`-Klasse. Diese Klasse stellt mehrere Methoden bereit um Statements hinzuzufügen:

```
void addStatement (string $graphUri, string $subject, string $predicate,
string $object, [-array $options = array()], [ $useAcl = true]);
```

Diese Methode fügt ein einzelnes Statement zur unter `$graphUri` angegebenen Wissensbasis/Graphen hinzu. Dabei repräsentieren `subject`, `$predicate` und `$object` ein Tripel. Optional können noch Optionen angegeben werden.

```
void addMultipleStatements ($graphUri, $statementsArray, [$useAc = true],
string $graphIri);
```

Funktioniert ähnlich wie die Methode `addStatement()`. Hier wird allerdings ein ganzes Array von Tripeln übergeben.

```
void addStatementFromObjects (string $modelIri, Erfurt_Rdf_Resource
$subject, Erfurt_Rdf_Resource $predicate, $object);
```

Im Gegensatz zu den ersten beiden Methoden, werden an dieser Stellen die Tripel nicht als Strings übergeben, sondern Subjekt und Prädikat sind vom Typ `Erfurt_Rdf_Resource`, wobei Object vom Typ `Erfurt_Rdf_Node` ist.

Alternativ kann man die Methoden `addStatement()` und `addMultipleStatements()` der Klasse `Erfurt_Rdf_Model` benutzen, welche die entsprechende `Erfurt_Store`-Methode aufruft. Hierbei entfällt das Angeben der Graph-Uri, da diese Methoden in `Erfurt_Rdf_Model` sich auf dem Graph beziehen, der von dieser Klasse repräsentiert wird.

1. Allgemeines

Die Aufgabe unseres Plugins ist es, die Möglichkeiten der Navigation durch das OntoWiki unabhängig des zu Grunde liegenden Modells deutlich zu erhöhen. Dies gelingt durch die sinnvolle Nutzung von Inferenzen aus den schon gespeicherten Daten. Neue implizite Informationen sollen dann über die Weboberfläche des Systems dem Benutzer zusätzlich, getrennt und hervorgehoben, angezeigt werden. Am Ende sollen zwei von einander getrennte Modelle als Basis der Informationsanzeige dienen.

2. Produktübersicht

Im Rahmen der Softwarestudie wollen wir uns mit dem System des Ontowiki und dessen Extension Struktur vertraut machen. Da es unsere Aufgabe und Ziel ist, ein EasyInference Plugin für das Ontowiki zu erstellen, wird unser Extension Typ ein Plugin sein. Dieser Prototyp soll dann exemplarisch die Inferenz-Aufgabe der Inverse lösen.

```
{?P @has owl:inverseOf ?Q. ?S ?P ?O} => {?O ?Q ?S}
```

Das Inferenz-Plugin wird im Ontowiki eingesetzt und steht ausschließlich dem Ontology-Engineer zur Verfügung. Der Prototyp generiert ein Inferenzmodell mit inversen Eigenschaften zu einer Ontowiki Wissensbasis. Aber auch andere Regeln werden später mit dem fertigen Produkt anwendbar sein.

3. Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Die grundlegenden Prinzipien für das Programmieren einer Erweiterung des Ontowiki sind schon in der eigentlichen Programmierung der Hauptsoftware vorgegeben. Auf Basis von PHP und dem MVC-Framework Zend stellt uns das Ontowiki eine Reihe von nützlichen Klassen und Methoden zur Verfügung, wobei eine strikte Objektorientierung von Zend vorgegeben ist. Also Datenbank-Grundlage dient uns im Moment Virtuoso, wobei auch MySQL als Data-Storage in Frage käme.

Das Gesamtsystem wurde von uns in kleinere Module zerlegt. Das Modul dient dabei als Behälter für Funktionen oder Zuständigkeiten des Systems. Die Zerlegung des Systems in Module erleichtert die Entwicklung, Änderung und Wartung des Systems – vor allem behält man dadurch den Überblick. Die Implementationsarbeiten konnten dadurch auch leicht auf die einzelnen Teammitglieder verteilt werden.

4. Grundsätzliche Struktur- und Entwurfsprinzen der einzelnen Pakete

Der Prototyp besteht aktuell aus sieben Hauptmethoden, welche durch zwei Hilfsmethoden ergänzt werden. Außerdem definieren wir für den klasseninteren Zugriff drei Klassenvariablen. Im folgenden wollen wir jede Klasse kurz vorstellen und dazu erklären, was für Funktionen dahinter stecken. Das Plugin selbst ist eine Klasse `InferencePlugin`, welche die Ontowiki Klasse `OntoWiki_Plugin` erweitert.

Klassenvariablen:

`$_owApp`	Ontowiki-Application Instanz
`$_erfurt`	Erfurt-Application Instanz

Methoden:

```
public function init();
```
An dieser Stelle wird das Inferenz-Plugin initialisiert. Es wird nach dem aktuellen Benutzer und dessen Status gefragt und überprüft, ob zum aktuell selektierten Modell bereits ein Inferenz-Modell existiert - falls nicht, wird ein neues erstellt.

```
private function hasInferenceModel($selectedModel);
```
Überprüft, ob das übergebene seleketierte Modell bereits ein Inferenz-Modell besitzt und gibt entsprechend `true` oder `false` als Rückgabewert zurück.

```
private function isInferenceModel($selectedModel);
```
Überprüft, ob das ausgewählte Modell eine Inferenz-Modell ist oder nicht und gibt dementsprechend `true` oder `false` als Rückgabewert zurück.

```
private function createInferenceModel($normalModel);
```
Diese Funktion erstellt ein inferiertes Modell auf Basis des übergebenen Modells. Dabei wird der Modell Iri um den String „`inference/`“erweitert. Danach wird das Inferenz-Modell initialisiert. Anschließend werden die inferierten Daten über einen Query abgefragt und ins Inferenz-Modell geschrieben. Dabei können auch mehrere Regeln beachtet werden.

```
private function initInferenceModel($model, $sourceModulname);
```
Diese Methode generiert ein Grundmodell als Basis für das Inferenz-Modell und schreibt ein paar vordefinierte Statements und Literale hinein.

```
private function getQueryFromRule($rule, $sourceModel);
```
Um aus einer vordefinierten Regel auch eine SPARQL-Anfrage zu generieren, ist es wichtig, diese mittels einiger Ersetzungsfunktionen zu generieren. Diese Methode erstellt aus einer Regel eine SPARQL-Anfrage um diese auf die aktuelle Wissensbasis anzuwenden. Der Rückgabewert ist ein Array, welches die Anfrage (Query), das Subjekt, Prädikat und das Objekt enthält.

```
private function replaceAllPrefixes($str, $prefix, $ns);
private function replaceAll($str, $replacement, $data);
```
Diese beiden Methoden dienen nur zur String-Konvertierung bei der Generierung der SPARQL-Anfragen aus den gegebenen Inferenz-Regeln. Sie haben nur wenig Bedeutung für da Plugin und wurden nur der Übersicht wegen, erstellt.

Zusammenfassung

Mittels der Softwarestudie ist es unserem Team gelungen, die Funktionalität des Ontowiki-Systems besser zu verstehen und darauf aufbauend einen eigenen kleinen Prototypen als Extension bzw. Plugin zu entwickeln. Dabei ist es uns gelungen, die aktuelle Wissensbasis auszulesen und anhand vorgegebener Inferenzregeln ein Inferenz-Modell zu erstellen. Dieses wurde dann als neue Wissensbasis dem Ontowiki-System hinzugefügt.

Die Aufgabe bestand lediglich darin, die einfache Regel der Inferenz zu implementieren. Wir haben unseren Prototypen schon so weit gebracht, dass er im Prinzip jede beliebige Regel in eine SPARQL-Anweisung umwandeln kann und somit auf des Modell anwenden kann. Im Moment sind diese Regeln allerdings noch hart im Quellcode integriert.

Dabei sind uns noch einige Probleme, besonders technischer Natur aufgefallen. Wir mussten zum Beispiel mit einem fehlerhaften Professorenkatalog kämpfen. Auf dieser Basis war es natürlich nicht möglich, ein fehlerfreies Inferenzmodell zu erstellen. Außerdem gibt es noch einige Beschränkungen vom PHP. Läuft ein Script beispielsweise länger als 120 Sekunden wird es abgebrochen. Auch dadurch können fehlerhafte Modelle entstehen.