Entwurfsbeschreibung

17.05.2010

1 Allgemeines

OntoWiki ist ein Online-Wissensmanagement-System, dessen Inhalte von den Nutzern nicht nur gelesen, sondern auch nach eigenen Vorstellungen geändert und ergänzt werden können. "Onto" bezieht sich dabei auf die semantische Struktur. Die Open-Source-Software OntoWiki ist also eine Plattform zur agilen Verwaltung von Semantic-Web-Wissensbasen und vermag Informationen komplex zu verknüpfen. Semantische Daten werden in unterschiedlichen Ansichten präsentiert und können intuitiv verändert werden.

2 Produktübersicht

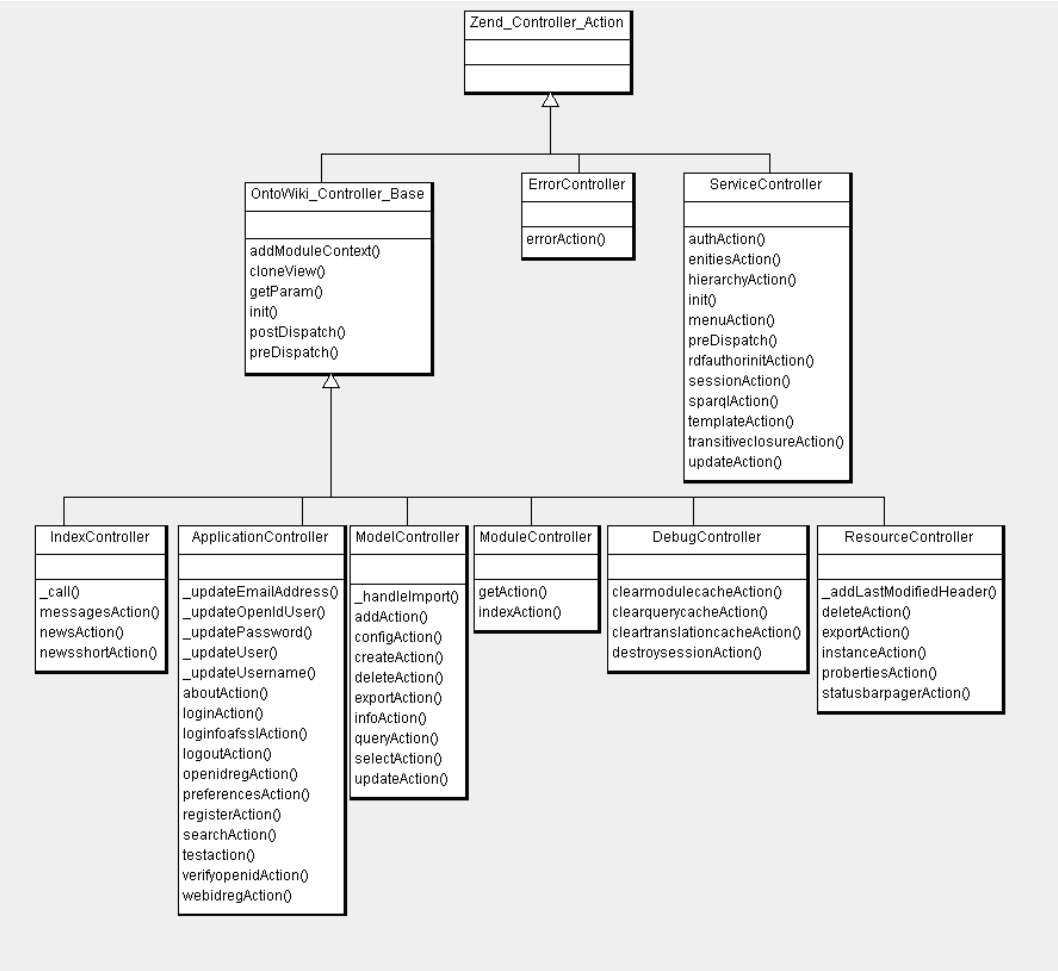
OntoWiki ermöglicht dem Nutzer Informationen in Wissenbasen zu sammeln, zu pflegen und zu bearbeiten. Hierbei wird nach dem Web 2.0-Prinzip der Nutzer aktiv eingebunden, indem er seine Anregungen und Vorschläge mit einbringt und diese diskutiert werden können. Das Arbeiten mit semantischen Inhalten erfolgt für den Nutzer intuitiv über einen RDF-Editor, ähnlich wie WYSIWYG für Textdokumente. Jegliche Veränderungen werden dokumentiert und können, falls notwendig, auch wieder rückgängig gemacht werden. Weiter bietet OntoWiki für den Nutzer die Möglichkeit einer semantischen Suche, wobei eine Volltextsuche nach den gegeben Literalen stattfindet. Die Ergebnismenge kann noch sortiert und gefiltert werden. Es werden Statistiken über die Popularität von Inhalten und die Aktivität des Nutzers geführt. Dadurch werden Inhalte in den Wissenbasen bezüglich ihrer Qualität, Präsentation und Originalität bewertet und auch aktive Nutzer, die zur Weiterentwicklung von Wissenbasen beitragen, können geehrt werden. Durch das einfache Einbinden von Plug-Ins kann OntoWiki den meisten

persönliche Bedürfnissen gerecht werden und somit um nahe zu beliebige Funktionalität erweitert werden. Auch die Nutzer werden an den Entwicklungsprozess aktiv durch die Community-Unterstützung mit eingebunden.

3 Struktur des Gesamtsystemes

3.1 Umsetzung der MVC-Architektur im Ontowiki

Im allgemeinen besteht die MVC-Architektur aus den drei Komponenten Model, View und Controller. Dabei enthält das Model die darzustellenden Daten, die View stellt die Daten aus dem Model dar und nimmt Benutzerinteraktionen entgegen und der Controller registriert die Bentzerinteraktionen und reagiert entsprechend darauf. Im Ontowiki wird diese Architektur in Verbindung mit Zend realisiert. So leitet sich die `OntoWiki_View` von der `Zend_View` und die `OntoWiki_Controller_Base` von `Zend_Controller_Action` ab. Die `OntoWiki_View` enthält grundlegende Methoden zur Erstellung und Darstellung der graphischen Benutzeroberfläche. Die Superklasse `OntoWiki_Controller_Base` beinhaltet verschiedene Controllermethoden wie z. B. `init`, `postDispatch`. Von dieser Klasse werden die Controller `ApplicationController`, `DebugController`, `IndexController`, `ModelController`, `ModuleController` und `ResourceController` abgeleitet. Der `Error Controller` und der `ServiceController` werden ebenfalls von der `Zend_Contoller_Action` abgeleitet.



Der `ModelController` enthält Funktionen zum Hinzufügen, Konfigurieren, Erstellen, Löschen und Exportieren von Ontologien. Der `ResourceController` verfügt über änliche Methoden wie der `ModelController`, wie zum Beipiel das Löschen und Exportieren sowie das Anzeigen von Resourcen und deren Eigenschaften. Der `ErrorController` enthält Default-Actions, die im Falle eines Fehlers bereitgestellt werden. Der `ServiceController` enthält die verschiedensten Methoden wie z. B. für die Suche von Entitys, für Hierarchie, für das Erzeugen von JSON für die Weiterführung von `OntoWiki_Menu`, für die JSON Ausgabe vom versteckten RDF Autor Auswahlordner, der RDF Autor Anfangskonfiguration und der transitiven Schließung von Resourcen zu einer gegebenen Startresource, für die Implementierung des SPARQL Protokolls, für das Rendern von Templates und für Updates. In OntoWiki existiert keine eigene Model-Klasse, da die zu präsentierenden Daten im Wissensbasis-Format vorliegen und mittel Erfurt-Framwork manipuliert wer-

den. Für Plug-Ins bietet OntoWiki die Möglichkeit durch einen spezifischen Controller, welcher auf eine eigene View verweist, die Datendarstellung zu manipulieren. Durch die eigens implementierten Controller kann die Sicht nach Bedarf angepasst werden.

3.2 Die Rolle von Zend in OntoWiki

Das Zend Framework stellt eine objenktorientierte Webaplikation dar, welche über eine grundlegene MVC-Architektur Implementation verfügt. Diese wird, wie bereits beschieben, in OntoWiki u.a. durch die Klassen `OntoWiki_Controller_Base` genutzt bzw. daraus abgeleitet und um zusätzliche Funktionen ergänzt. Die `OntoWiki_Bootstrap.php` wird ebenfalls von Zend abgeleitet (`Zend_Application_Bootstrap_Bootstrap`), und erzeugt eine Instanz des `FrontControllers` (Zend), wobei u.a. die Controller registriert werden. Weiterhin bietet das Zend Framework einen Datenbankadapter, wodurch je nach Bedarf OntoWiki statt mit Virtuoso, dessen Unterstützung ohne Zend implementiert ist, auch mit MySQL oder anderen, von ZendDB unterstützten, RDBMS betrieben werden kann.

3.3 Die Strukturierung der Packages und APIs

Ein Package dient der Zusammenfassung logisch zusammengehöriger Klassen. Dies wird im Dokumentationscode mit dem Tag `@package` gekennzeichnet, kann aber auch über die angelegte Ordnerstruktur der Klassen vorgenommen werden.

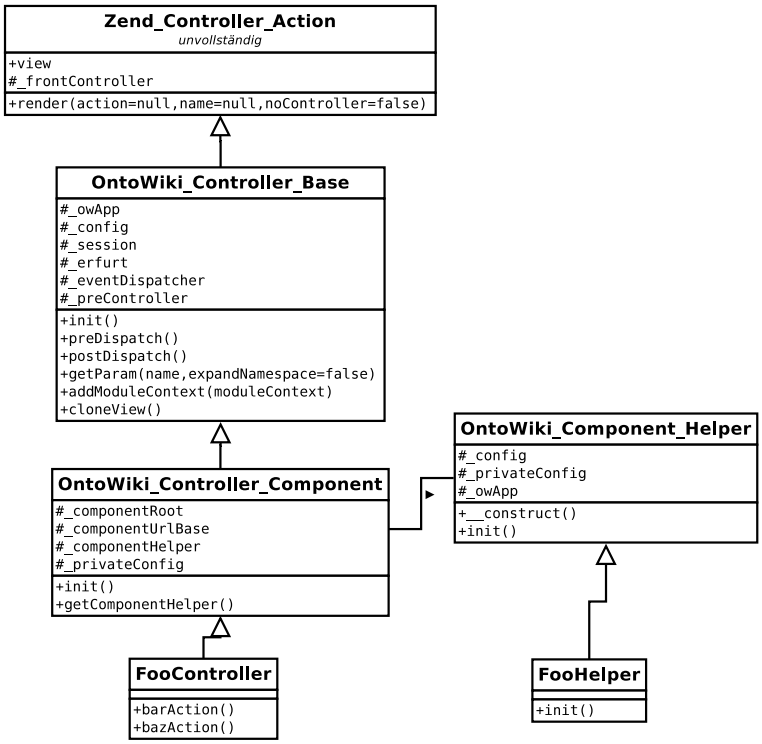
4 Struktur der Einzelkomponenten

Die Klassen im Ordner `applications` setzen die Kernfunktionen von OntoWiki um. Im `libraries`-Ordner befinden sich die von OntoWiki benötigten Bibliotheken: Das Erfurt Semantic Web Framework zur Arbeit mit Ontologien, das Zend-Framework für die Anwendungsarchitektur (insbesondere das MVC-Muster) und zum Datenbankzugriff, ein Mime-Parser sowie mit `RDFauthor` ein in JavaScript geschriebenes Werkzeug zum komfortablen Bearbeiten der Properties im Browser. Der Ordner `extensions` bietet Platz für austauschbare Module. Die Paketierung erfolgt dabei über `@package`-Direktiven sowie eine festgelegte Ordnerstruktur und Dateinamenskonventionen, wobei sich jede Erweiterung in einem eigenen Ordner befindet und einen eigenen Paketnamen erhält. Bei Developer Extensions bestimmen außerdem Vorhandensein und Inhalt der Datei

`component.ini`, `module.ini` bzw. `plugin.ini`, ob die Erweiterung aktiv ist und wie sie sich verhält.

4.1 Developer Extensions

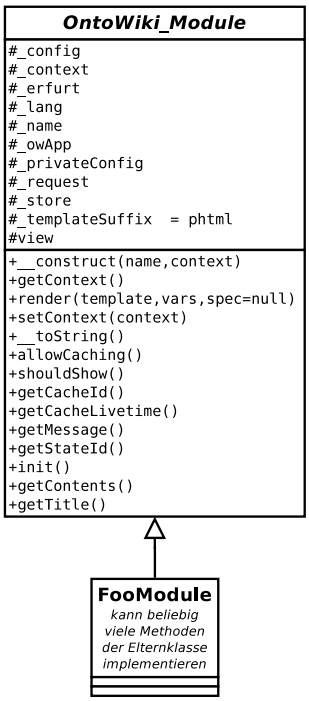
4.1.1 Komponenten (Unterordner: components)



Komponenten sind von `OntoWiki_Controller_Component` abgeleitete Controller im Sinne des MVC-Patterns, die auf bestimmte Aktionen reagieren können. Die Klasse `Foo Controller` sollte in der Datei `FooController.php` liegen. Außerdem kann in der Datei `FooHelper.php` eine Klasse `FooHelper` definiert werden, deren `init`-Funktion bei jedem Seitenaufruf ausgeführt wird. Der relative Teil einer OntoWiki-URL hat meist die Form `controller/action/r/ressourceName`. Für den Controller `foo` wird dabei die Komponente `foo` gesucht und für die Action `bar` die Funktion `barAction()` des `FooControllers` ausgeführt. In der `barAction()` ist es z.B. möglich, das Objekt `$this ->view` zu mit Daten zu füllen, welche welche im PHP-Skript `bar.phtml` dann als `$this` verfügbar sind. Die Anzeige (View) des Tab-Inhalts übernimmt die Datei `templates`

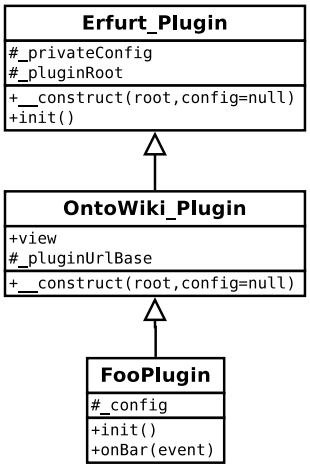
`/foo/bar.phtml`. Die Klasse `Zend_Controller_Action` kümmert sich darum, dass deren Inhalt evaluiert wird. Ein Beispiel für eine Komponente ist die `source`-Komponente, die einen Tab zum Bearbeiten des Quelltextes zur Verfügung stellt.

4.1.2 Modul (Unterordner: modules)



Von der abstrakten Klasse `OntoWiki_Module` abgeleitete Module können eigene Unterfenster zur Verfügung stellen. Sie können dynamisch konfiguriert werden, indem Werte aus der Datei `module.ini` durch Rückgabewerte entsprechender redefinierter Methoden überschrieben werden. Die Namenskonvention sieht die Schreibweise `FooModule` vor. Ein Beispiel für ein Modul ist das `Login`-Modul, welches das Login-Unterfenster am linken Rand anzeigt.

4.1.3 Plugins (Unterordner: plugins)



Von `OntoWiki_Plugin` abgeleitete Plugins können mit beliebigem Code auf beliebige Ereignisse reagieren. Die Definition der Klasse `FooPlugin` sollte sich dabei in der Datei `foo.php` im Ordner `foo` befinden.

Ein Beispiel ist das `mailtolink`-Plugin, welches E-Mail-Adressen automatisch durch mailto-Links ersetzt.

4.2 Non-Developer Extensions

Außerdem gibt es im `extensions`-Verzeichnis Unterordner Themen (`themes`) und Übersetzungen (`translations`).

4.3 Implementierung eigener Sichten

Frage: Wie kann das Projekt durch z.B. die Implementierung eigener Sichten erweitert werden? Beschreiben Sie dies anhand des Modells.

- mit einer Komponente mit entsprechender `action.phtml`
- mit einer neuen Action in einer bestehenden Komponente