Editor für die Entwicklung IT-basierter Dienstleistungen

Designbeschreibung

Version 1.0

Gliederung:

1. Allgemeines

Das Produkt stellt eine webfähige Entwicklung der IT-basierten Dienstleistungen dar. Dabei soll der Kunde mit Hilfe des Produkts in die Lage versetzt werden, auf der Basis einer zugrunde liegenden Beschreibung des Komponentenmodells (Metamodells) einzelne Komponenten und deren Komposition zu modellieren. Dabei soll das Zusammenspiel zwischen den funktionalen und nichtfunktionalen Aspekten berücksichtigt werden. Der Kunde soll mit dem Editor die komplette Dienstleistung entwerfen und deren einzelnen Komponenten, Modelle und Diagramme erstellen und bearbeiten können.

Das System basiert auf der Java-Servlet-Technologie. Das System benötigt einen Servlet-fähiger Webserver, auf dem die erforderliche Software installiert ist, sowie ein JavaScriptfähiger Client.

Wir wählen den XML-Standard für die Datenspeicherung, weil es für System günstig ist, und Dateien leicht in andere Formate überführt werden können.

2. Produktübersicht

Der Editor für die Entwicklung IT-basierter Dienstleistungen ist eine webbasierte Software, die Projektverwaltung ermöglicht. Die Hauptfunktionen sind Erzeugen, Löschen, Speichern und Bearbeiten des aktuellen Projekts, sowie seiner Modelle und Komponenten (Abbildung 1) .

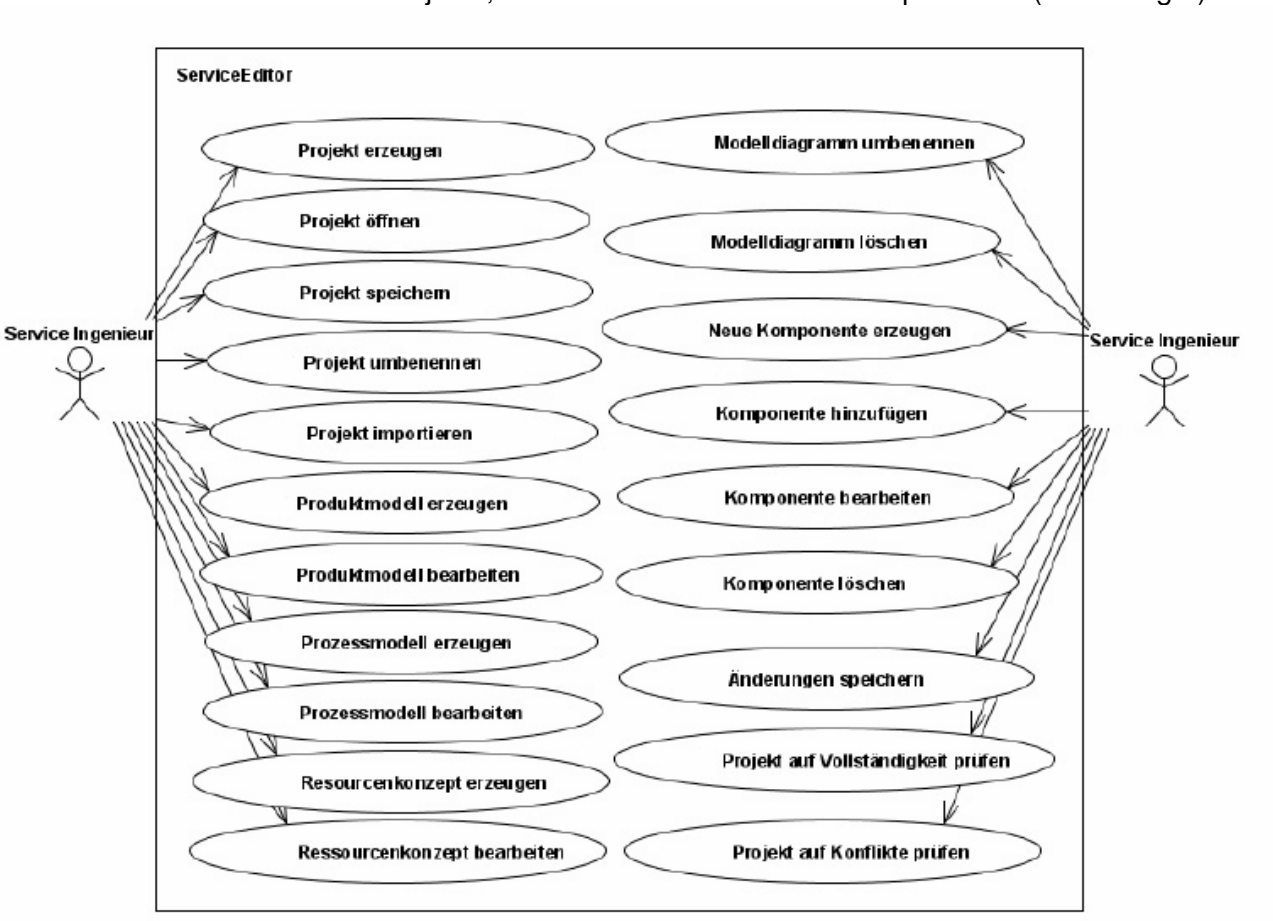

Abbildung 1 Produktübersicht

3. Grundsätzliche Designentscheidungen

3.1 MVC

Der Editor für die Entwicklung IT-basierter Dienstleistungen folgt dem *Model-View-Controller*-Konzept und soll deswegen sehr modular aufgebaut sein. Das *Modell* realisiert die Kernfunktionalität, die *View* liefert die Schnittstelle für die Bildschirmpräsentation, der *Controller* stellt die Schnittstelle für die Benutzereingaben dar. Ziel dieser Architektur ist die Trennung der Verarbeitung eines Problems (*Model*) von dessen Darstellung (*View*) und von der Manipulation (*Controller*) der Anwendungsdaten durch Benutzereingaben. Die Hauptschichten dabei sind: Persistenzschicht(Model), Präsentationsschicht(View) und Geschäftslogikschicht(Controller). Somit besteht das System aus vier Untersystemen: dem Paket Modul, dem Paket Komponente, dem Paket Editor und dem Paket View.

3.1 Servlet

Wir haben entschieden, unseren Editor als Servlet zu programmieren. Erstens soll unsere Software webbasiert sein, ein Servlet, installiert auf dem Server, nimmt die Anfragen von Clients entgegen und beantwortet sie. Zweitens, kann der Inhalt der Antworten dynamisch sein. Das bedeutet, er entspricht dem aktuellen Zustand der Daten, die auf dem Server liegen.

3.2 Apache Tomcat

Apache tomcat liefert uns eine Umgebung zur Ausführung von Java-Code auf einem Webserver. Es ist ein in Java geschriebener Servlet-Container, der mittels JSP-Compilers Jasper JavaServer Pages in Servlets übersetzen und ausführen kann.

3.3 Struts- Framework

Es gibt viele Möglichkeiten, das *MVC*-Konzept zu implementieren. Eine von denen ist das Struts-Framework. Erstens, kann man mit Struts fast sofort seine Web-Applikationen entwickeln, auch wenn man nicht viel Zeit und nur wenig Erfahrung hat. Zweitens, erlaubt eine gute Dokumentation dem Anfänger relativ schnell mit dem Framework vertraut zu werden.

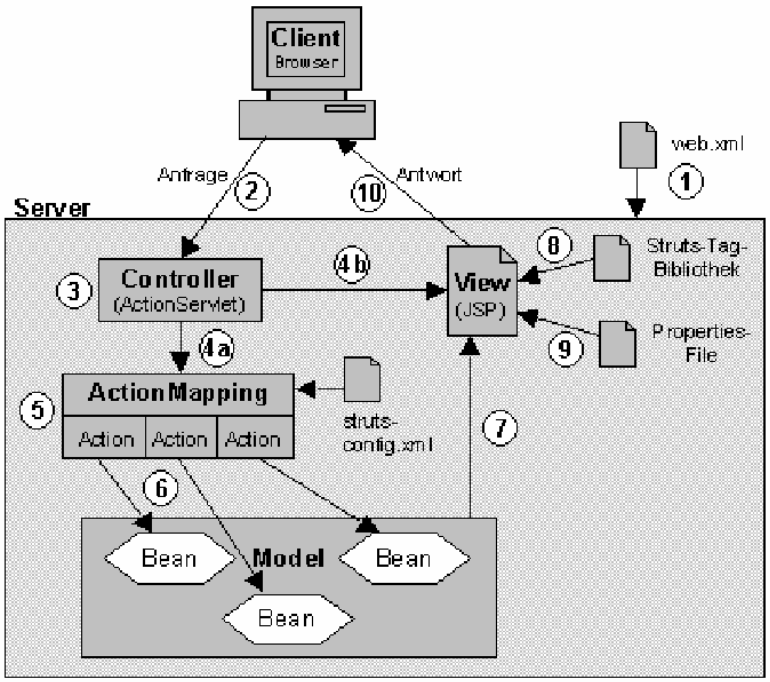

Abbildung 2 Programmablauf in Struts

Der erste Schritt im Ablauf ist nicht die Anfrage des Browsers, sondern das Setup der Struts-Komponenten (1). Der Web-Server konfiguriert sie mit Hilfe der Datei web.xml (siehe Kapitel 5.3.1 *Konfiguration des Frameworks*). Ist das Setup abgeschlossen, wird auf eine Anfrage (http-Request) des Benutzers gewartet. Wird vom Benutzer eine Seite abgefragt oder ein Formular an den Server geschickt, wird ein Request ausgelöst (2). Diese Anfrage wird vom Controller entgegengenommen (3). Dieser entscheidet, ob er die Anfrage an die Klasse ActionMapping übergibt (4a) oder ob er die Anfrage gleich an die View-Komponente weiterleiten kann (4b). Dies erkennt er an der Tatsache, ob Geschäftslogik ausgeführt werden muss oder ob direkt eine Ausgabe erfolgen kann. Wird eine bestimmte Action-Klasse anhand des Mappings in der Datei struts-config.xml zur Verarbeitung ausgewählt (5), leitet diese Klasse die Anfrage an die

entsprechende JavaBean, welche die Geschäftslogik enthält, weiter (6). Die verschiedenen JavaBeans stellen das Model dar. Nach Abarbeitung der Geschäftslogik, wird das Ergebnis an die JavaServer Page, die als View-Komponente dient, übergeben (7). Die Struts-Tag-Bibliothek definiert die verschiedenen Struts-spezifischen Tags, die in der JSP verwendet werden (8). Das Properties-File (9) enthält die Texte der jeweiligen Sprache, die in der JSP angewandt wird. Schließlich wird die generierte Seite mit den Ergebnisdaten an den Browser zurückgesendet (10).

3.4 Dynamisches Modell

Beschreibung eines typischen Anwendungsfalls :

Die Bearbeitung eines Geschäftsprozesses erfolgt in mehreren Schritten. Der Browser sendet einen Request an den Webserver (z.B. Tomcat), welcher einen Servlet-Container bereitstellt. Der Container ruft demnach die Methode doGet (bzw. doPost) des von Struts vorgegebenen ActionServlets aus, welcher die Anfrage parst und entsprechende Bearbeitungsvorgänge anstößt. Die Zuordnung zwischen der URL und der zuständigen Action-Klasse wird dabei in der Datei *struts-config.xml* angegeben.

Die Abbildung 3 Sequenzdiagramm stellt ein Beispiel für die Geschäftsvorgänge bei der Erstellung eines Projekts dar. Dabei wird zunächst ein leeres Projekt angelegt. Im zweiten Schritt erfolgt das Hinzufügen eines Produktmodells, welches dann mit Komponenten bzw. Diagrammen konkretisiert werden kann.

Für die Erstellung eines Projekts ist die *ProjektCreator*-Instanz zuständig, diese ruft die Methode *newProject* des *Editor*-Objekts auf, welche ein leeres *Project*-Objekt erzeugt und es mit Namen und Erstellungsdatum versieht. Das *Project* wird an den ProjektCreator zürückgegeben, welcher ihn zwischen zwei Requests (z.B. in der Session) merkt.

Im nächsten Vorgang muss dem Projekt ein Modell hinzugefügt werden. Dies geschieht, indem der Benutzer eine entsprechende Anfrage an den Webserver sendet. Die Anfrage wird wieder an den *ActionServlet* weitergeleitet, welcher nun die *execute*-Methode der *ProjecktSaver*-Instanz aufruft. Danach instantiiert der *ProjectSaver* ein *Produktmodell* (in diesem Anwendungsfall) und fügt diesem neue Komponenten (Objekte des Typs *ProdKomponente* (in diesem *Anwendungsfall*)) sowie ein Modelldiagramm (*ModellDiagramm*-Instanz) hinzu. Schließlich wird das Modell dem Projekt mittels der Methode *addModul* hinzugefügt. Als letztes wird das Projekt bzw. sein Metamodell über die *saveProject*-Methode gespeichert sowie ein *HttpResponse* für den Benutzer generiert.

Sequenzdiagramm:

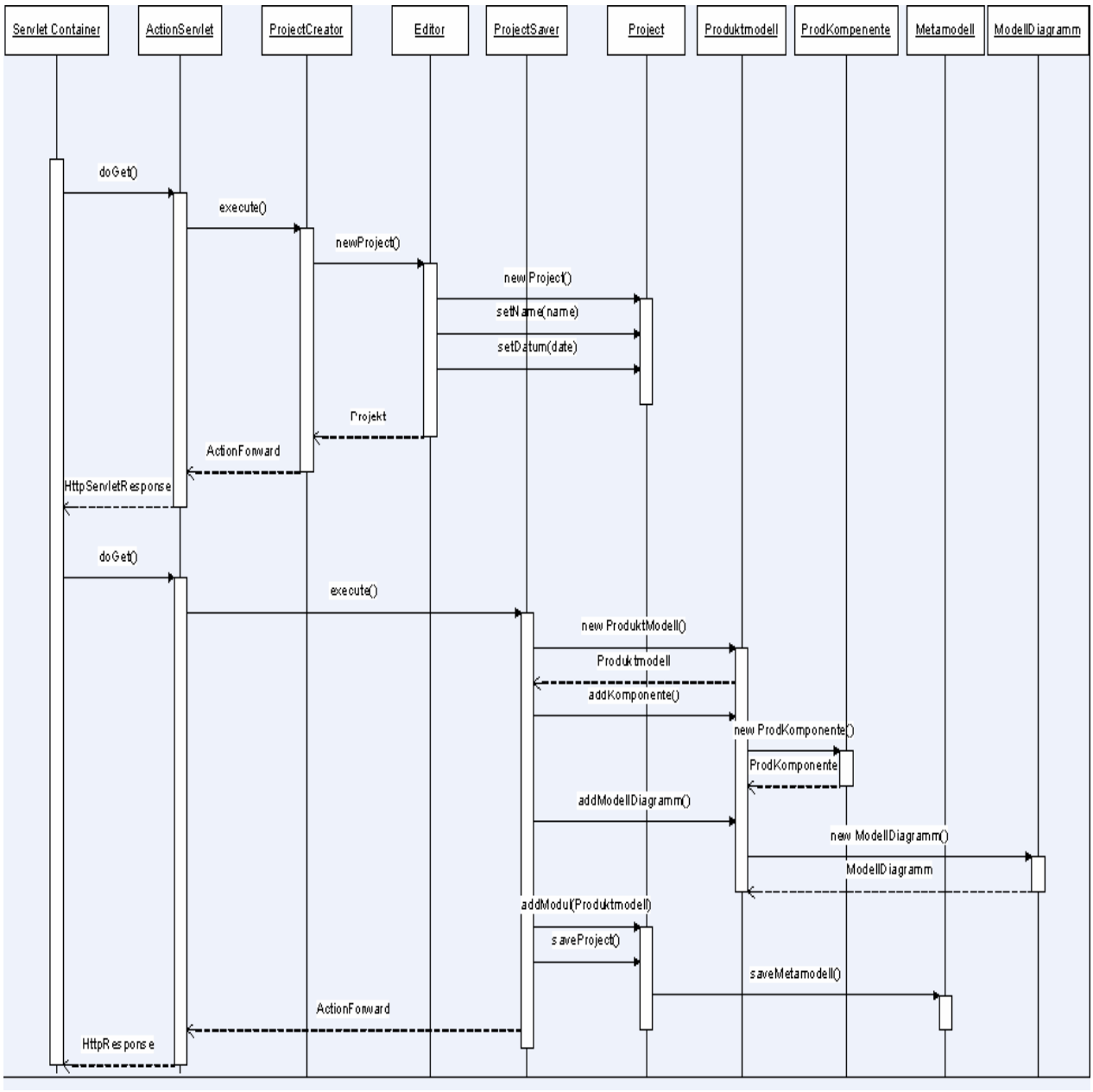

Abbildung 3 Sequenzdiagramm

4. Paket- und Klassenstruktur

Paketstruktur:

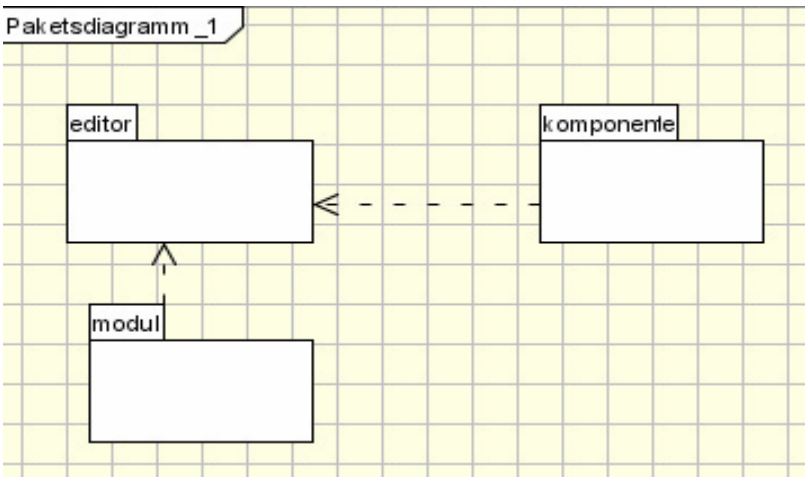

Abbildung 4 Paketstruktur

Klassenstruktur:

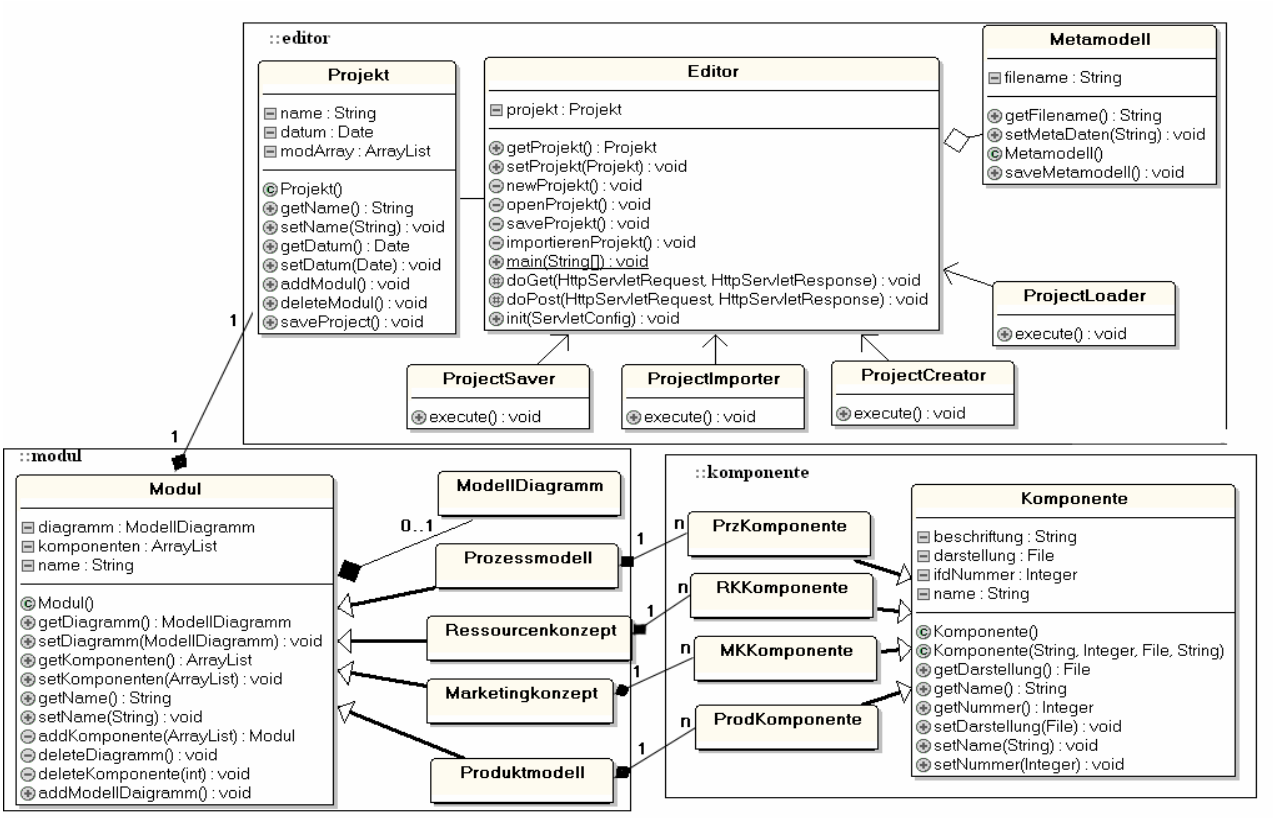

Abbildung 5 Klassenstruktur

Es gibt drei Pakete in unserem System:

- Package **editor**

Die Hauptklasse im Package *editor* ist die Klasse Editor. Diese Klasse ist für die Übersetzung der Benutzereingaben verantwortlich. Dabei kann man ein leeres bzw. neues Projekt angelegen oder importieren. Nach der Bearbeitung eines Projekts, kann man es speichern oder umbenennen. Im zweiten Schritt erfolgt das Hinzufügen eines Produktmodells.

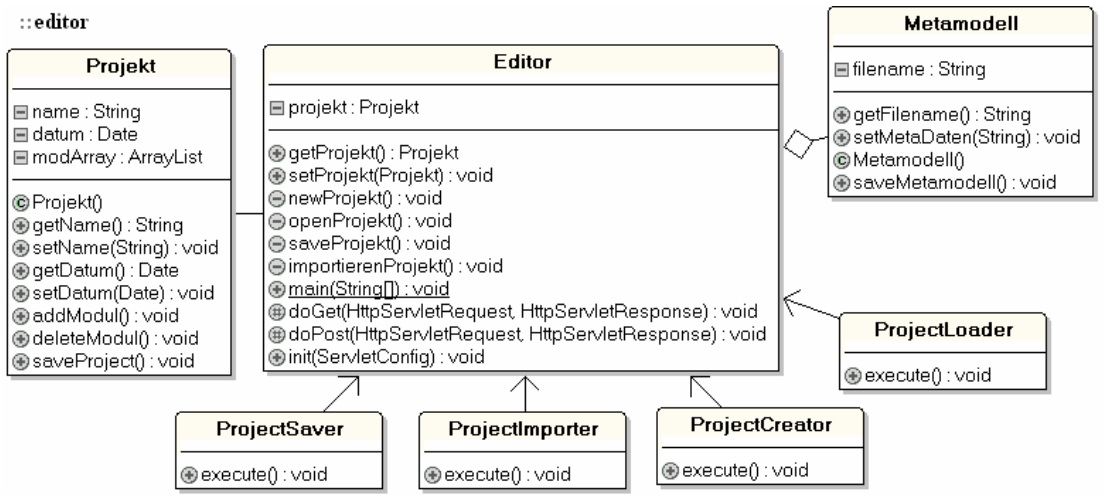

Abbildung 6 package editor

- Package **modul**

Die Hauptklasse im Package *modul* ist die abstrakte Klasse Modul. Es wird nach Abfrage ein Prozessmodell, Ressourcenkonzept, Marketingkonzept oder Produktmodell und das dazu gehörige Modelldiagramm erstellt.

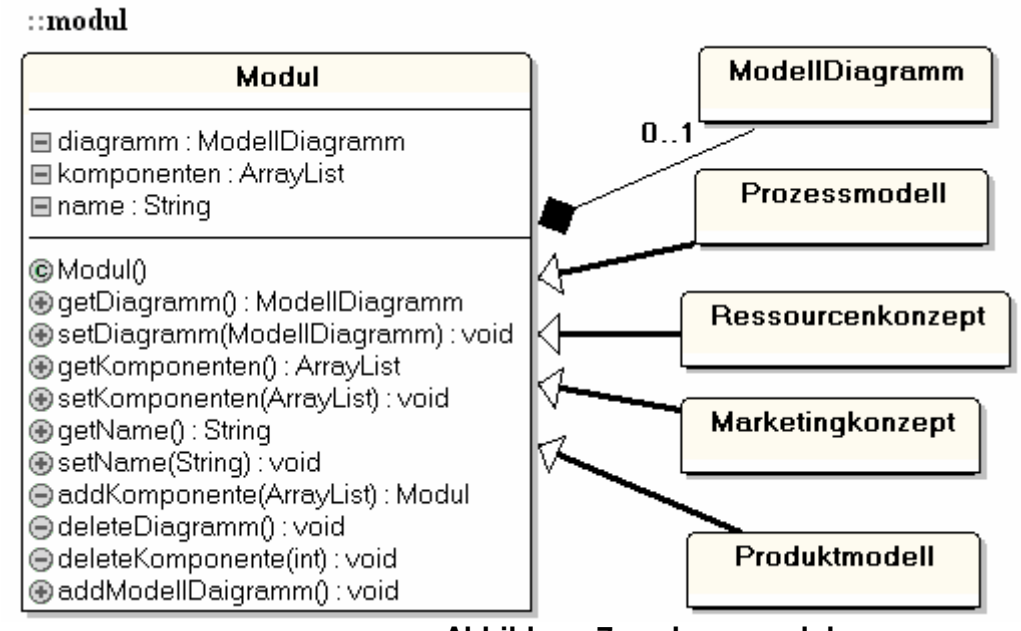

Abbildung 7 package modul

- Package **komponente**

Die Hauptklasse im Package *komponente* ist die Klasse Komponente. Nach dem Starten des Editors, Öffnen des Projektes und des entsprechenden Modells kann eine neue erzeugte Komponente dem Modell hinzugefügt werden.

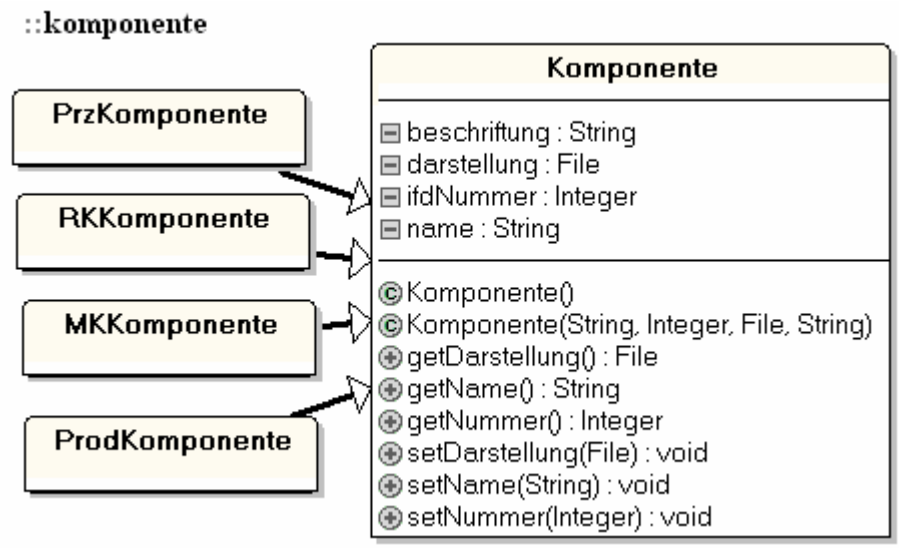

Abbildung 8 package komponente