Bachelorarbeit

Oryx Dokumentation

Willi Tscheschner

Betreuer
Prof. Mathias Weske, Hasso-Plattner-Institute, Potsdam, Germany
Hagen Overdick, Hasso-Plattner-Institute, Potsdam, Germany
Gero Decker, Hasso-Plattner-Institute, Potsdam, Germany

30. Juni 2007

Zusammenfassung

Diese Bachelorarbeit ist Teil einer Dokumentation für das Bachelorprojekt B7 - „Browser-basierter Geschäftsprozess Editor“ vom Fachbereich „Business Process Technology“ des Hasso-Plattner Institut in Potsdam im Studienjahr 2006/07. Ziel des Bachelorprojektes ist es, eine Web-Anwendung zu realisieren, mit dem Geschäftsprozesse modelliert werden können. Als Notationssprache wurde Business Process Modeling Notation (BPMN) [1] genutzt.

Der Schwerpunkt der vorliegenden Bachelorarbeit ist eine genaue Spezifikation des Editors und die Lieferung von wesentlichen Implementierungsdetails. Sie dient somit als Erstliteratur für Weiterentwicklungen am Editor.

Abstract

This final Bachelor's paper is part of the documentation of the Bachelor project B7 - „Browser-based Business Process Editor“ of the department „Business Process Technology“ at the Hasso-Plattner Institute in Potsdam, Germany in the year 2006/2007. The Goal of the project was to realize a web application for business process modelling. The used process notation is the Business Process Modeling Notation (BPMN) [1].

This final bachelor's paper focuses on describing the concepts used for the application and implementation details. It provides the necessary information for further development.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Potsdam, 30. Juni 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Bachelorarbeit ist Teil einer Dokumentation für das Bachelorprojekt B7 - „Browser-basierter Geschäftsprozess Editor“ vom Fachbereich „Business Process Technology“ des Hasso-Plattner Institut in Potsdam im Studienjahr 2006/07. Als Synonym für den Projektnamen steht Oryx.

Geschäftsprozesse sind Arbeitsabläufe im wirtschaftlichen Sinn. Sie können weitere Geschäftsprozesse beinhalten oder andere anstoßen. Prozessmanagement beschäftigt sich mit der Identifikation, Dokumentation, Steuerung oder Verbesserung von Geschäftsprozessen.
Ziel des Projektes Oryx ist es, ein Programm zu realisieren, mit dem Geschäftsprozesse modelliert werden können. Als Notationssprache wurde Business Process Modeling Notation (BPMN) [1] genutzt. Dies ist eine Prozessmodellierungssprache, die häufig in der Wirtschaft Anwendung findet.

Bei der Realisierung des Editors gab es verschiedene Aspekte, die berücksichtigt wurden.

Der Editor wurde als web-basierte Anwendung realisiert. Damit entfällt somit die aufwendige Installation der Software und Prozesse können weltweit modelliert und editiert werden.
Eine Zwischenschicht erlaubt die Abstraktion der Daten zum Editor hin. So kommuniziert der Editor nur mit dieser Schicht, um an Daten zu gelangen oder diese zu manipulieren. Die Architektur der Zwischenschicht basiert auf dem Konzept von Representational State Transfer (REST) [5]. Dies sagt aus, dass nur durch die Verben 'GET', 'PUT', 'POST' und 'DELETE' Einfluss auf eine Ressource im Web genommen werden kann - also das Erhalten einer Repräsentation, das Neuanlegen, das Verändern und das Löschen einer Ressource. Eine Ressource kann alles sein, was eine Identität im World Wide Web (WWW) besitzt.

Ein weiterer Aspekt des Editors sind die einfachen und flexiblen Schnittstellen, die es einerseits ermöglichen die Funktionalität des Editors zu erweitern und andererseits dem Editor neue Prozessmodellierungssprachen hinzuzufügen. Durch die Verwendung von Stencil Sets, einer Beschreibung der grafischen Prozesselemente, ihrer

Eigenschaften und Attribute, kann dem Editor sehr einfach das Wissen und das Regelwerk von Prozesssprachen übergeben werden.
Durch die Umsetzung eines Plugin-Konzeptes ist es möglich die grafische Benutzeroberfläche anzupassen, neue Funktionen zu erstellen oder vorhandene Funktionalität zu erweitern.

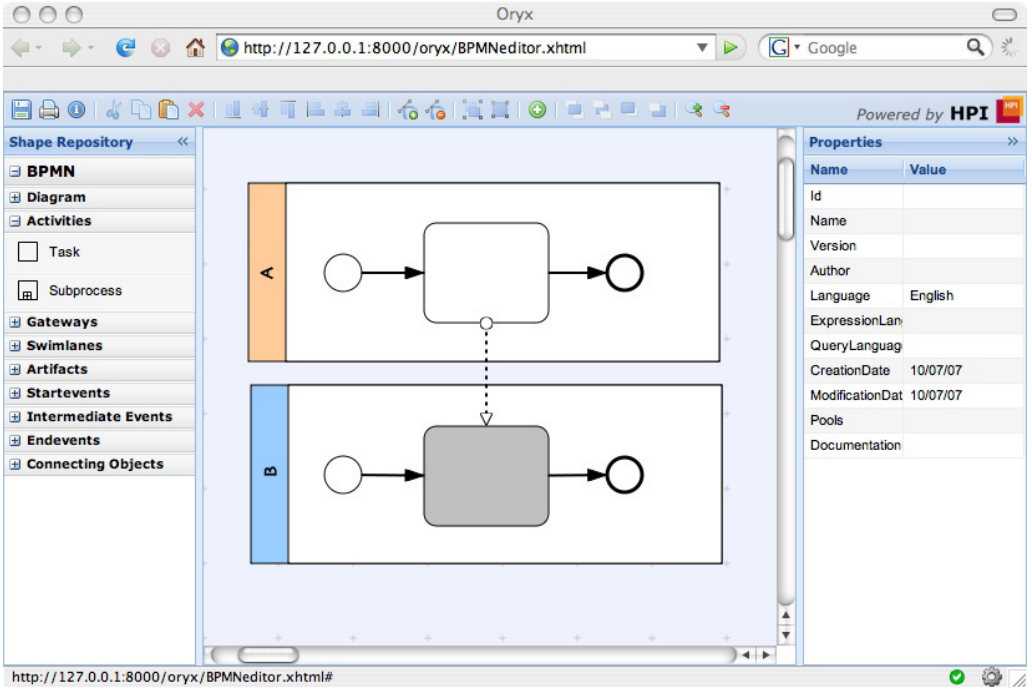

Abbildung 1.1: Editor - Screenshot

Die Datenhaltung und die Serverimplementierung ist Teil des Partnerprojektes B8 - Thanis. Diese implementieren einen Server, der die Prozessausführung realisiert. So können Prozesse auf einfache Art und Weise ausgeführt werden. Ziel ist die Verknüpfung beider Projekte zu einer Anwendung. So kann man Geschäftsprozesse mit dem Editor modellieren und mit dem Server abwickeln. Zusätzlich zum Server stellt das andere Projekt ein Front-End zur Verfügung, das während der Prozessausführung angezeigt wird. In diesem können Daten für die Prozessausführung hinterlegt werden.

1.1 Übersicht

Der Schwerpunkt der vorliegenden Bachelorarbeit ist eine genaue Spezifikation des Editors und die Lieferung von wesentlichen Implementierungsdetails. Sie dient somit als Erstliteratur für Weiterentwicklungen am Editor.
Diese Arbeit ist analog zu einem klassischen Entwicklungsprozess aufgebaut. Es werden zu Beginn die Anforderungen beschrieben, die in der ersten Projektphase spezifiziert wurden. Danach werden einige Details über die zugrundeliegende Technologie erläutert. Der Schwerpunkt der Arbeit liegt in der Beschreibung der Architektur und der Darstellung unserer Implementierung. Abschließend werden Details über die Plugins und mögliche Erweiterungen des Editors erläutert.

Diese Arbeit bildet die Grundlage für weitere Studien, Implementierungen oder Erweiterungen am Editor.

Neben dieser Dokumentation gibt es noch drei weitere Arbeiten, die sich mit dem Editor und dem Projekt Oryx beschäftigen: „Embedding Business Process Data into the Web“, „Stencil Set Specification“ und „BPMN Stencil Set Implementation“. In der Bachelorarbeit „Embedding Business Process Data into the Web“ von Martin Czuchra [4] wird die Kommunikation mit dem Server beschrieben. In dieser wird genauer spezifiziert, wie die Daten im Document Object Model (DOM) stehen und wie die Synchronisierung aller Anwendungen auf der Seite realisiert ist.
Nicolas Peters erläutert in seiner Arbeit „Stencil Set Specification“ [11], wie der Aufbau eines Stencil Sets aussieht, welche Eigenschaften und Abhängigkeiten es gibt und wie der Editor die Daten richtig interpretieren und übernehmen kann.
In der dritten Bachelorarbeit „BPMN Stencil Set Implementation“ wird von Daniel Polak eine konkrete Umsetzung eines BPMN Stencil Sets dargestellt [12]. In dieser wird die Spezifikation des Stencil Sets überprüft und am Beispiel von BPMN umgesetzt.

2. Analyse

In diesem Kapitel werden die Anforderungen an den Editor beschrieben.

2.1 Anforderungen

Die Anforderungen sind gestellte Ziele an das Projekt und den Editor, die am Anfang des Projektes von uns spezifiziert wurden. Sie beschreiben den Zustand den das System am Ende erreichen soll. Sie sind somit Rahmenbedingungen, die teilweise obligatorisch und teilweise optional sind.

2.1.1 REST basierte Schnittstelle

Das Laden und Speichern der Modelle auf dem Server ist über eine Representational State Transfer (REST) basierende Schnittstelle realisiert.
Eine Zwischenschicht zum Lesen und Schreiben auf dem XHTML-DOM existiert.
Anmerkung: Diese Zwischenschicht dient der Synchronisation mit dem Server.

2.1.2 Integration in eine XHTML-Seite

Der Editor ist in eine XHTML-Seite eingebunden.
Alle benötigten Dateien werden dynamisch vom Server nachgeladen.

2.1.3 W3C Standard

Der Editor erzeugt Daten, die konform zu den Standards vom World Wide Web Consortium (W3C)[1] sind.
Anmerkung: Es ist somit wichtig, dass alle SVG-, XHTML-, CSS-, RDF- oder andere Daten in einer W3C gültigen Syntax beschrieben werden.

2.1.4 Lauffähig in allen Browsern

Der Editor ist im Firefox 2.0 lauffähig.
(optional) Der Editor läuft in allen Browsern und ist jeweils daraufhin optimiert.

[1] Link: http://www.w3.org/

2.1.5 Stencil Set Spezifikation

Es existiert eine Spezifikation, die ein Stencil Set beschreibt. In dieser ist festgehalten, wie Stencils beschrieben sind und welche Besonderheiten beachtet werden müssen.
Anmerkung: Ein Stencil ist die graphische Repräsentation, die die eigentlichen Prozesselemente realisiert. Das Stencil Set ist somit die Beschreibung der Prozesssprache durch Stencils, inklusive des Regelwerks und zusätzlicher Eigenschaften.

2.1.6 Unterstützung von BPMN

Der Editor bringt ein komplettes BPMN Stencil Set mit.
Anmerkung: Es dient vor allem dazu, die Funktionsvielfalt des Editors zu demonstrieren und zu überprüfen, ob die Stencil Set Spezifikation korrekt und vollständig beschrieben ist. Diese Umsetzung ist Teil der Bachelorarbeit von Daniel Polak [12].

2.1.7 Plugin API

Es ist ein Plugin-Konzept umgesetzt. Der Editor bietet eine Schnittstelle zu den Plugins an, über die die komplette Funktionalität des Editors bereitgestellt wird.
(optional) Ein Plugin kann sich an jedes Event des Editors registrieren.
(optional) Plugins definieren eigene Events, die wiederum von anderen Plugins verwendet werden.

2.1.8 Desktop feeling

Der Editor ist wie eine Desktop-Anwendung konzipiert.
Er setzt die Konzepte von IBM - „Design Basics“ [3] um.
Der Editor kann komplett mit der Maus gesteuert werden.
Anmerkung: Somit wird ein besserer Bedienkomfort für unerfahrene Benutzer garantiert.

2.1.9 Benutzerschnittstelle

Der Editor realisiert folgende Benutzerfunktionalität:

Drag und Drop

Der Benutzer hat die Möglichkeit, einzelne Objekte innerhalb des Editors von einer Position zur einer anderen zu verschieben.

Delete, Copy and Paste

Es ist möglich, Objekte im Editor zu löschen, zu kopieren oder auszuschneiden, um sie an einer anderen Position wieder einzufügen.
Anmerkung: Beim Einfügen wird darauf geachtet, dass alle Attribute und dessen Werte kopiert und übernommen werden.

Resize

Objekte besitzen die Möglichkeit die Größe zu verändern.
Anmerkung: Jedes Objekt enthält die Information darüber, wie es die Größe ändern kann. So ist die minimale und die maximale Größe angegeben und es ist gesetzt, ob es horizontal oder vertikal skalierbar ist.

Alignment

Objekte richten sich an anderen Objekten im Editor aus.
Anmerkung: Es kann ausgewählt werden, an welchen Objekten sie sich ausrichten. Die Ausrichtung erfolgt einerseits auf der Z-Achse, andererseits an der Position anderer Objekte. Bei der Ausrichtung der Position gibt es folgende Möglichkeiten: oben, unten, links und rechts und jeweils die Mitte im horizontalen und vertikalen. In der Z-Ebene gibt es folgende Möglichkeiten: nach ganz vorn, eine Ebene nach vorn, eine Ebene nach hinten und nach ganz hinten.

Ausrichten am Grid

Objekte im Editor richten sich an einem Grid, also an einem Gitter aus Punkten, aus. Objekte im Editor richten sich an anderen Objekten im Editor aus.

2.1.10 Shape Repository

Der Editor bietet ein Shape Repository an.
Anmerkung: Im Shape Repository werden alle Elemente der geladenen Prozessmodellierungssprachen angezeigt und können dem aktuellen Prozess, mittels Drag und Drop, hinzugefügt werden. Sie sind nach Prozesssprachen gruppiert.

2.1.11 Vorhandene Prozesse

(optional) Vorhandene Prozesse werden im Prozess Repository angezeigt.
Anmerkung: Diese können im aktuellen Prozess als Subprozess eingefügt werden.
(optional) Der Editor bietet eine Suchfunktion für vorhandene Prozesse an, die auf dem Server verfügbar sind.

2.1.12 Hinzufügen von neuen Prozesssprachen

(optional) Es ist möglich dem Editor während der Ausführungszeit neue Prozessmodellierungssprachen hinzuzufügen.

2.1.13 Versionierung und Collaboration

(optional) Der Editor bietet Versionierung und Collaboration an.
Anmerkung: Da die Umsetzung dieses Punktes hauptsächlich vom Server getragen wird, stellt der Editor, sofern dieses vom Server unterstützt wird, Benutzerschnittstellen dafür bereit.

2.1.14 Sichten auf ein Prozess-Modell

Der Editor stellt verschiedene Sichten auf einen Prozess bereit.
Anmerkung: Es ist möglich, unterschiedliche Arten von Sichten zu speichern und wieder zu öffnen. Dies ist wichtig, da verschiedene Teilnehmer an einem Prozess verschiedene Sichten und Berechtigungen haben.

2.1.15 Modell Personalisierung

(optional) Es ist möglich die vorhandenen Modelle optisch anzupassen, solange es die Spezifikation der Prozessmodellierungssprache erlaubt. So kann die Hintergundfarbe und die Linienfarbe angepasst werden.
Anmerkung: Dadurch sind die Modelle besser lesbar und der Informationsgehalt kann erweitert werden. Diese Umsetzung ist Teil der Stencil Set Spezifikation die in der Bachelorarbeit von Nicolas Peters [11] beschrieben wird.

2.1.16 UI Personalisierung

(optional) Der Editor bietet die Möglichkeiten an, die Benutzeroberfläche an den jeweiligen Benutzer anzupassen.
(optional) Die Einstellungen dafür werden gespeichert und können wieder geladen werden.
Anmerkung: Dadurch wird eine besser Benutzbarkeit des Editors erzielt.

2.1.17 Kontext sensitives Menü

Es wird ein Kontext sensitives Menu bereitgestellt.
Anmerkung: Dadurch kommt der Benutzer schneller an die Funktionalität, die gerade benötigt wird.

2.1.18 Drucken

Der aktuelle Prozess kann auf einer DIN-A4 Seite ausgedruckt werden.

2.1.19 Exportfunktion

(optional) Der aktuelle Prozess kann nach folgenden Formaten exportiert werden: PDF, SVG, Raster-Grafik, XPDL.

3. Technologien

In diesem Kapitel wird für den Editor eine geeignete Technologie ausgewählt und eine kleine Einführung in die ausgewählten Technologien gegeben.

3.1 Technologieentscheidung

Bei der Technologieauswahl standen folgende Kriterien im Vordergrund:

- Browserunterstützung
- Verfügbarkeit
- Datenstruktur
- Verfügbare Entwicklungsumgebungen
- Verfügbare Bibliotheken

Dabei werden folgende Technologien untersucht:

Scalable Vector Graphics (SVG) ist ein XML-Datenformat zur Beschreibung von Vektorgrafik. Sie ist durch das W3C standardisiert und in Version 1.1 verfügbar [8].

Adobe Flex [10] ist eine in Action-Script und MXML geschriebene Programmiersprache, die zur Laufzeit Adobe Flash erzeugt. Mit Adobe Flex werden verschiedene Bibliotheken mitgeliefert. Unter anderem auch eine Bibliothek für Vektor-Grafik.

Canvas-Tag ist eine HTML-Erweiterung. Sie ist standardisiert von der Web Hypertext Application Technology Working Group (WHATWG) und ist im Web Application 1.0 Working Draft [9] beschrieben. Es soll voraussichtlich in den XHTML-Standard 2.0 eingebunden werden. Das Canvas-Tag ist eine Zeichenfläche, die eine API für JavaScript anbietet, mit der auf dieser gezeichnet werden kann.

Als Entwicklungsumgebung wurden ein konventioneller Text-Editor, Adobe Flex-Builder und Google Web Toolkit (GWT) verwendet.

3.1.1 Browserunterstützung

In der nachstehenden Tabelle 3.1 wird kurz beschrieben, welcher Browser welche Technologien unterstützen.

	SVG	Adobe Flex	Canvas
IE 7.0	Unterstützung mittels Adobe SVG Viewer	ab Flash Player 9.0	Unterstützung mittels Plugin
Firefox 2.0	Unterstützt (nicht implementiert sind Animation, Filter und Teile des Text/Font-Moduls [6])	Ab Flash Player 9.0	Unterstützt
Safari 2	Keine Unterstützung (ab Safari 3 soll es native unterstützt werden)	Ab Flash Player 9.0	Unterstützt
Opera	Unterstützt (Teile des Text/Font-Moduls sind nicht implementiert [2])	Ab Flash Player 9.0	Unterstützt

Abbildung 3.1: Browserunterstützung

3.1.2 Verfügbarkeit

SVG ist von dem W3C standardisiert und ist ein offener Standard. Somit ist es frei verfügbar.

Adobe Flex ist eine proprietäre Programmiersprache, die von Adobe (ehemals Macromedia) entwickelt wurde. Sie ist frei verfügbar, wobei es teilweise Einschränkungen bei den verschiedenen Bibliotheken gibt. So ist z.B. die Servernutzung mittels Adobe Flex kostenpflichtig. Der Compiler ist hingegen kostenlos und wird von Adobe bereitgestellt.

Das Canvas-Tag ist eine durch WHATWG standardisierte Erweiterung des HTML-Standards und ist frei verfügbar.

3.1.3 Datenstruktur

SVG liegt in einer XML-Struktur vor und wird zur Laufzeit vom Browser interpretiert.

Adobe Flex liegt zur Entwicklungszeit als XML-Struktur vor, was dann zur Laufzeit als Flash kompiliert wird. Somit liegt im Browser Binär-Code vor, welcher nur durch ein Plugin interpretiert werden kann.

Das Canvas-Tag ist Bestandteil von HTML. Mittels JavaScript können auf diesem Tag Zeichenoperationen ausgeführt werden. Es liegen dadurch zur Ausführungszeit alle Objekte nur in JavaScript vor und nicht im DOM der Seite.

3.1.4 Verfügbare Entwicklungsumgebungen

Für SVG und das Canvas-Tag werden eine Reihe von Editoren und Entwicklungsumgebungen kostenlos angeboten; so z.B. Aptana, ein auf Eclipse-basierte Entwicklungsumgebung für Web-Anwendungen. Im Gegensatz dazu ist die Entwicklungsumgebung für Adobe Flex, Adobe FlexBuilder, kostenpflichtig und vergleichbares gibt es nicht. Somit würde hier nur der Texteditor in Frage kommen.

3.1.5 Verfügbare Bibliotheken

Für SVG oder das Canvas-Tag sind nur sehr wenige Bibliotheken verfügbar. Allerdings gibt es für HTML (das von beiden genutzt wird) eine Menge von Bibliotheken, welche man z.B. für die GUI oder etwaige Berechnungen nutzen kann. Für Adobe Flex hingegen gibt es eine Reihe von Bibliotheken, die für die Umsetzung der GUI oder das Zeichnen von Vektorgraphiken genutzt werden können.

3.1.6 Fazit der Evaluierung

SVG ist frei verfügbar, wird stetig weiterentwickelt und ist vom W3C standardisiert. Es ist einfach und intuitiv zu lesen und zu verstehen. Die grafischen Prozesselemente können durch eine deklarative Beschreibungssprache erstellt und modelliert werden, was die Erstellung eines Stencil Sets sehr vereinfachen würde. SVG und JavaScript sind in fast allen Browsern verfügbar und können ohne Plugins interpretiert werden. JavaScript ist sehr verständlich und kann schnell erlernt werden.

Adobe Flex kann nur durch ein Plugin im Browser interpretiert werden, da im Klient nur Binär-Code vorliegt. Es ist proprietär, wobei es auf ActionScript aufsetzt. Des Weiteren ist nur eine kostenpflichtige Entwicklungsumgebung verfügbar.
Beim Canvas-Tag ist es ähnlich. Es ist zur Zeit nicht vom W3C standardisiert und alle grafischen Elemente sind nur in JavaScript-Objekten verfügbar.
Bei beiden wäre nur schwer eine deklarative Beschreibung der Stencils möglich, was die Beschreibung eines Stencil Sets sehr schwierig macht.

Nachdem einige Prototypen gebaut wurden (siehe Ordner 'evalutation', 4.4) um einen noch genaueren Einblick in die Technologien zu bekommen, wurden SVG und JavaScript als Technologie ausgewählt. Der Editor soll somit in einer XHTML-Seite eingebettet werden, was auch die Anforderung 2.1.2 realisiert. Dadurch dass nur vom W3C standardisierte Technologien genutzt werden, wird auch die Anforderung 2.1.3 umgesetzt.

In der ersten Version steht die Lauffähigkeit im Firefox 2.0 im Vordergrund (somit wird die Anforderung 2.1.4 abgedeckt).

3.2 JavaScript

JavaScript ist eine dynamisch typisierte, objektbasierte Skriptsprache, die prototypisch ist.
Prototypsich heißt, dass ein Datentyp als Vorlage eines neuen Datentyps verwendet wird; so nutzt man z.B. 'Object' als Prototyp für 'Array'.
In JavaScript gibt es dadurch nur vier wesentliche Datentypen und zwar 'Object' als Basis-Objekt für weiter Datentypen, 'String' als alphanumerischer Datentyp, 'Number' als numerischer Datentyp und 'Bool' als boolscher Datentyp. Daneben gibt es

noch die Datentypen 'null' und 'undefined', die auch als leeren String, '0' oder 'false' interpretiert werden können. Von 'Object' abgeleitete Datentypen sind somit 'Function', 'Array' oder 'Hash'.

Des Weiteren sind alle Objekte in JavaScript dynamisch typisierte, d.h. Typkonvertierung geschieht automatisch. Trotz alledem kann aber noch auf den Typ referenziert werden. So ist z.B. folgendes möglich:

```
var a = true    // -> true
alert(typeof a) //    -> boolean
a = 1 + a       // -> 2   (da, true := 1)
alert(typeof a) //    -> number
a = "1" + a     // -> "12"
alert(typeof a) //    -> string
```

In JavaScript gibt es mehrere Möglichkeiten eine Funktion zu realisieren:

```
function Add(a,b){ return a + b };

var Add = function(a,b){ return a + b };

var Add = new Function("a", "b", " return a + b ");
   // Hierbei sind die Argumente als String angegeben

Add(1,2);            //-> 3
```

Da in JavaScript eine Funktion auch nur ein 'Object' ist, kann dies auch als Eigenschaft gespeichert werden.

```
var Add = function(a,b){ return a + b };

var Mathe = {};
Mathe.add = Add;

Mathe.add(1,2);      // -> 3
```

Weitere Informationen zu JavaScript, inkl. einer einfachen und gut verständlichen Beschreibung der Objekte und der DOM API, befindet sich auf den Seiten von SelfHTML [1].

Einige wichtige Objekte in JavaScript werden in der Tabelle 3.2 beschrieben.

[1] Link: http://de.selfhtml.org/javascript/index.htm

Objekt	Beschreibung
window	Referenz auf das aktuelle Fenster. Darüber können Eigenschaften des Fenster, wie z.B. Größe oder Position, geändert werden. Dialogboxen sind verfügbar und Timeouts können gesetzt werden.
document	Referenz auf das aktuelle Document Object Model (DOM). Darüber kann auf einen einzelnen Knoten referenziert oder neue Knoten erzeugt werden.
node	Referenz auf einen einzelnen Knoten im DOM. Es bietet die Möglichkeit auf Folge-, Kind- oder Eltern-Knoten zu referenzieren, oder lesend und schreibend auf diesem Knoten zu operieren.
event	Bei der Registrierung auf ein Event bekommt man das aktuelle event-Objekt als Argument übergeben. Diese beinhaltet die kompletten Informationen, die das Event zur Verfügung stellt(z.B. Maus-Position oder Key-Code).
Math	Einfach mathematische Berechnungen, wie z.B. Runden, Exponieren, Sinus oder Kosinus.

Abbildung 3.2: JavaScript Objekte

Des Weiteren läuft JavaScript in einer sogenannten Sandbox im Browser. So kann nur auf den Browser Referenz genommen werden, nicht aber auf das umliegende System. Dadurch wird verhindert, dass bösartiger Code aus dem Internet auf dem System ausgeführt wird.
Das größte Problem bei JavaScript ist, dass es Single-Threaded ausgeführt wird. D.h. es läuft im Prozess des Editors und ihm wird nur ein Thread zugewiesen. Dies hat als Folge, dass erst am Ende eines Event-Durchlaufs die Seite gerendert werden kann und somit Web-Anwendungen bei langen Event-Durchläufen sehr inperfomant sind. Ein Trick, so etwas zu umgehen, wäre, dass ein Timeout für eine bestimmte Methode gesetzt wird, die zeitlich unspezifiziert ausgeführt werden kann. Somit würde diese an das Ende der Event-Schlange gesetzt und es könnte zwischendurch die Seite neu gezeichnet werden.

```
window.setTimeout(function(){ ... }.bind(this), 10);
```

3.3 SVG

SVG ist eine vom W3C standardisierte [8] Beschreibungssprache für Vektorgrafiken in XML.

Mittels sogenannten XML-Tags und Attributen kann eine vektorbasierte Grafik aufgebaut werden. Ein Pfad dient dabei als Basiselement. Aufbauend auf diesem gibt es noch weitere grafische Objekte, wie Rechteck, Kreis, Ellipse, Linie, Polylinie und Polygon, die jeweils spezifische Attribute besitzen. So kann z.B. die Füllung oder der Rand angepasst werden.

```
<path  d="M20 200 C20 150 200 250 200 200"
    style="fill:none; stroke-width:1; stroke:red;" />

<rect  x="10" y="10" width="100" height="30"
    rx="15" ry="15" style="stroke: red; fill: none;"/>
```

Abbildung 3.3: SVG

3.4 SVG Unterstützung im Firefox

SVG wird von Firefox 2.0 unterstützt. Allerdings ist die komplette SVG Spezifikation nicht umgesetzt. So wurde vom Firefox 2.0 die komplette Animation, SVG-Filter und Teile der Font-Spezifikation nicht implementiert. Diese sind allerdings für den Editor nicht relevant. Eine komplette Liste der unterstützten Elemente befindet sich im Mozilla Developer Center [6].

Das Einbinden bzw. Darstellen von SVG kann auf verschiedene Weise geschehen. Einerseits unterstützt der Firefox 2.0 das Anzeigen von reinen SVG-Dokumenten, andererseits unterstützt er das Darstellen von SVG in XHTML-Seiten. Hier ist es aber sehr wichtig die korrekten Namensräume von XHTML anzugeben.

Im Folgenden werden einige spezifische Fehler von SVG im Firefox 2.0 aufgelistet:

- Im Firefox 2.0 unter Mac OS X existiert eine Fehldarstellung von Schriften. So wird der Text nicht richtig gerendert, der in einer XHTML-Seite eingebunden ist [2]. D.h. die Textfarbe, also das 'Fill', wird nicht oder nur teilweise angezeigt. Was allerdings angezeigt wird, ist der Rahmen des Textes - also das 'Stroke'.
- Einige JavaScript-Funktionen für SVG-Elemente sind nicht implementiert. Es ist z.B. 'getIntersectionList' nicht implementiert, die für Schnittpunktberechnung sehr hilfreich wäre.
- Die Bounding-Box von SVG-Elementen kann nur abgefragt werden, wenn diese im DOM gerendert sind [3].

[2] Link: https://bugzilla.mozilla.org/show_bug.cgi?id=317759
[3] Link: https://bugzilla.mozilla.org/show_bug.cgi?id=293581

4. Entwurf und Implementierung des Editors

In diesem Kapitel wird der Editor im Einzelnen beschrieben. Es werden Konzepte genannt und anschließend wird auf die konkrete Umsetzung und Implementierung eingegangen.

4.1 Konzepte des Editors

Bei den Konzepten ist die Rede von Objekten im Editor. Dabei handelt es sich jeweils um ein graphisches Objekt auf der Zeichenfläche des Editors, dass ein Prozessmodellelement repräsentiert.

4.1.1 Bounds

Jedem Objekt wird eine Bounds zugewiesen. Eine Bounds ist die rechtwinkliche Fläche, die zwischen zwei Punkten liegt. Sie gibt die größtmögliche Ausdehnung eines Objektes an. Es gibt die Möglichkeit die Punkte neu anzugeben, bzw. sie absolut oder relativ zum vorherigen zu verschieben. Wenn die Bounds geändert wird, wird ein Flag gesetzt, welches signalisiert, dass das Objekt neu gezeichnet werden soll. Außerdem gibt es einen Callback-Mechanismus, an dem sich Objekte registrieren können, um zu erfahren, wann die Bounds sich geändert hat.

4.1.2 Kanten/Knoten

Der Editor unterscheidet bei den grafischen Objekten zwischen Kanten und Knoten. Beide Konstrukte werden durch ein Stencil Set beschrieben und mittels einer SVG-Repräsentation geladen. In den SVG-Dateien stehen zum einen die graphischen Repräsentationen und zum anderen Oryx-spezifische Einstellungen. So werden in diesen z.B. die Anzahl und Positionen der Docker und Magnets angegeben oder wie ein Stencil skaliert werden darf. Die genaue Beschreibung der Stencil Set Spezifikation wird von Nicolas Peters in seiner Bachelorarbeit beschrieben [11].

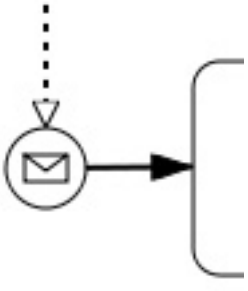

Abbildung 4.1: Kanten und Knoten

4.1.3 Refresh

Jedes Objekt hat eine Update und eine Refresh-Methode. In der Update-Methode wird, wenn die Bounds geändert wurde, die Refresh-Methode aufgerufen. Zusätzlich werden alle Update-Methoden der Kind-Elemente gerufen. So kann sichergestellt werden, dass jedes Kind-Element auch immer aktuell nach der Bounds gezeichnet wird. In der Refresh-Methode wird nun jedes Objekt nach der aktuelle Bounds neu ausgerichtet und gezeichnet. Sie ist somit ausschließlich für die Darstellung und das Neuzeichnen der SVG-Repräsentation an der Bounds zuständig.

4.1.4 Docker und Magnets

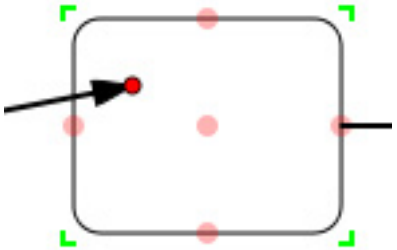

Abbildung 4.2: Docker und Magneten

Dockers sind Referenzpunkte für Kanten oder Knoten, die sich jeweils an ein anderes Objekt andocken können. Kanten haben mindestens zwei Docker (Anfang und Ende) und Knoten maximal einen (im Zentrum). In der Stencil Set Beschreibung ist spezifiziert, ob ein Docker sich an das Element andocken darf oder nicht. Zusätzlich gibt es Magneten, welche als Snap-Punkte für die Docker dienen. Sie können verwendet werden, um sicher zu stellen, dass bestimmte Referenzpunkte auf dem Objekt auch getroffen werden.

4.1.5 Plugin

Die Editorfunktionalität kann vollständig über Plugins abgerufen und erweitert werden. Dabei bekommt jedes Plugin bei der Initialisierung ein Interface vom Editor. Zwei Arten von Plugins können realisiert werden: Plugins, die die Funktionalität anbieten und welche, die vorhandene Plugins nutzen, um z.B. Funktionalität dem Benutzer grafisch anzuzeigen. Wenn ein Plugin Funktionalität implementiert, kann dies über eine Offer-Methode bei dem Editor registriert werden. Möchte ein Plugin hingegen auf vorhandene Plugins eingehen, muss eine spezielle Methode implementiert werden. Diese wird dann mit allen vorhandenen Plugindaten gerufen.

4.1.6 Selektion

Der Selektionsmechanismus ist Teil der Grundfunktionalität des Editors. Er stellt sicher, welche Elemente der aktuellen Selektion angehören und welche nicht. Eine Selektion ist eine Auswahl an Prozesselementen im Editor, die auf der Zeichenfläche liegen. Um auf Selektionsänderung reagieren zu können, gibt es ein Event, auf das man sich registrieren kann.

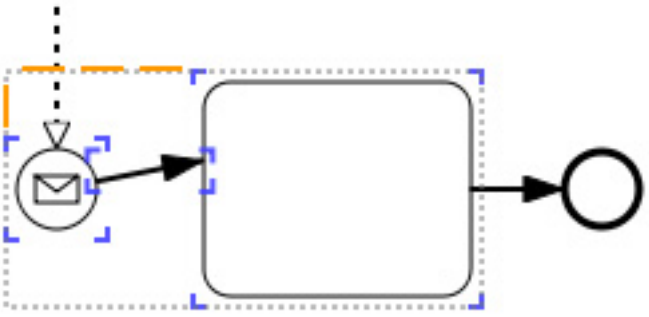

Abbildung 4.3: Selektion

4.2 Architektur des Editors

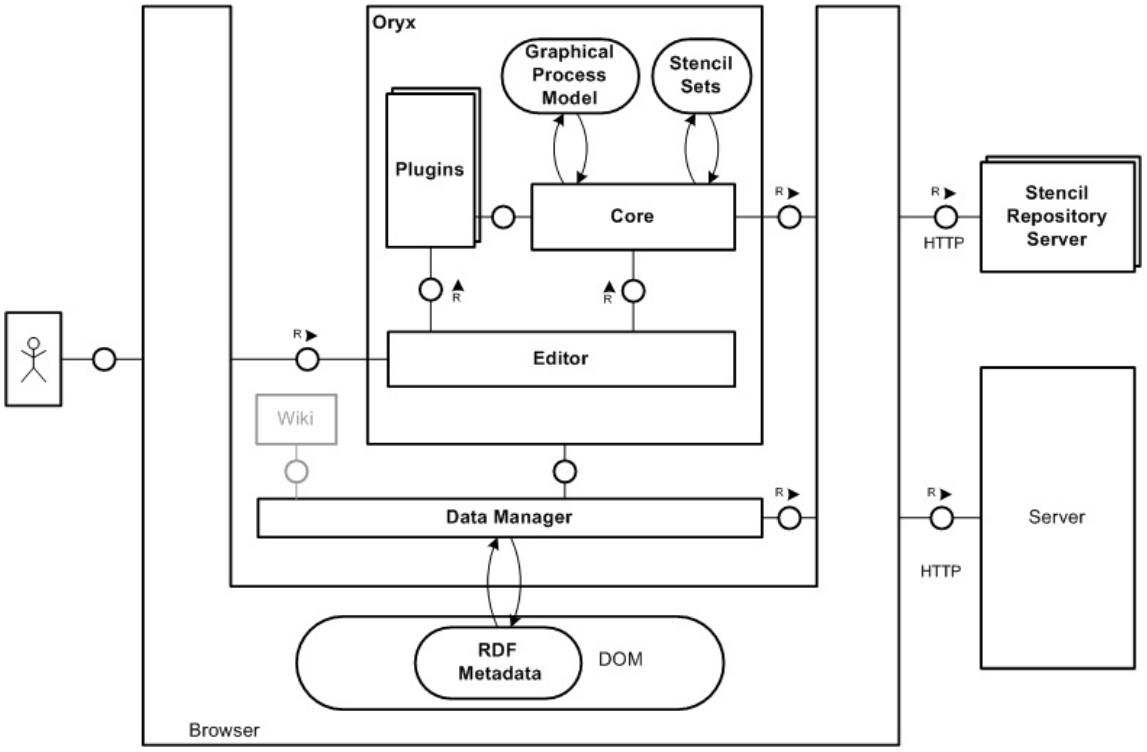

Abbildung 4.4: Architektur - Diagramm

Oryx ist in einer Browser-Umgebung eingebettet, in der auch die Daten liegen mit denen der Editor arbeitet. Den Zugriff auf die Daten, sowie die Persistierung regelt der Data Manager. Er stellt eine eigenständige Komponente dar, die als Datenschicht eine Abstraktionsebene unter dem Editor einzuordnen ist, wie die Abbildung 4.4 zeigt.

Die Komponente Oryx ist in drei wesentliche Bereiche aufgeteilt. In der Komponente Core ist die Kernfunktionalität des Editors implementiert, wie die Umsetzung von grafischen Objekten oder die Verarbeitung von Stencil Sets. Eine Reihe von Plugins implementieren zusätzliche Funktionalität wie Drag & Drop oder grafische Benutzerschnittstellen. Als letzte Komponente ist der Editor für die Initialisierung aller benötigten Objekte, sowie für die Abwicklung der Kommunikation zwischen den Komponenten zuständig.

Die Stencil Sets werden erst zur Laufzeit vom Stencil Repository Server nachgeladen.

Data Manager

Der Data Manager ist für den Zugriff und die Persistierung der Prozessdaten verantwortlich. Er stellt eine eigene Schicht in der Architektur dar, die auch von anderen Anwendungen benutzt werden kann. Er kapselt somit den Zugriff auf die Daten und stellt eine spezielle API dafür bereit. Der Data Manager realisiert somit die Anforderung 2.1.1.
Die Daten sind in der aktuell geöffneten Seite mittels eRDF eingebettet. Das Resource Description Framework (RDF) ist eine formale Sprache für die Beschreibung

von maschinenlesbaren Informationen im Internet. Eine Ressource im Allgemeinen kann alles sein, was einen eindeutigen Bezeichner (z.B. eine URI) besitzt. Das RDF beschreibt dieses mittels einer Tripel-Darstellung: Subjekt, Prädikat und Objekt. Dadurch wird also geklärt, was wie wohin refernziert. Embedded RDF (eRDF) ist ein Teilbereich von RDF, welches speziell für XHTML-Seiten konzipiert wurde. Eine genaue Beschreibung über eRDF und dem Data Manager befindet sich in der Bachelorarbeit von Martin Czuchra [4].

Core

Der wichtigste Teil der Core Komponente ist die Umsetzung von grafischen Objekten, die intern SVG für die grafische Darstellung nutzen. Darunter fallen z.B. die Zeichenfläche oder die grafischen Elemente einer Modellierungssprache, aber auch Kontrollelemente wie Dockers und Magnets. Hierzu gehören auch Methoden zur Manipulation dieser grafischen Objekte.

Daneben gehört zu diesem Paket auch die Implementation der Stencil Set Spezifikation, also die API für den Zugriff auf Stencil Set Beschreibungen. Die Anforderung für eine Stencil Set Beschreibung aus 2.1.5 wird dadurch erfüllt.

Des Weiteren sind in dieser Komponente Klassen für die Kapselung des Zugriffs auf SVG Elemente und Implementierungen von mathematischen Berechnungen zu finden.

Plugins

Mit der Hilfe von Plugins werden zwei Dinge implementiert. Zum einen erweitern sie die Funktionalität der Anwendung und zum anderen setzen sie die grafische Benutzerschnittstelle um (siehe Anforderung 2.1.9). Der Editor bietet den Plugins hierzu eine definierte Schnittstelle auf die Kernfunktionen von Oryx an. Der Editor und die Core Komponente werden bewusst schlank gehalten. So wird möglichst viel Funktionalität über die Plugins realisiert. Diese implementieren z.B. das Ausrichten, Kopieren, sowie Gruppieren von Objekten, aber auch eine Toolbar, die eine Benutzerschnittstelle für die Funktionen anbietet.

Eine genaue Liste aller implementierten Plugins ist im Abschnitt 5.3 zu finden.

Editor

Die Editor Komponente ist für die Initialisierung der Zeichenfläche, der Plugins, der Stencil Sets und der Prozessmodelle verantwortlich. Den Plugins wird vom Editor eine Schnittstelle bereitgestellt, die es den Plugins ermöglicht, auf Kernfunktionen des Editors zuzugreifen. So lassen sich z.B. über diese Schnittstelle neue Modellelemente erzeugen oder vorhandene löschen. Auch der Zugriff auf einen Event-Handler oder die geladenen Stencil Sets ist über die Schnittstelle möglich. Somit wird die Anforderung für eine Plugin API 2.1.7 abgehandelt.

4.3 Bibliotheken

Der Editor benutzt folgende externe Bibliotheken:

4.3.1 Prototype

Version: 1.5.1 RC3

Homepage: http://www.prototypejs.org/

Zweck: Prototype erweitert die JavaScript-Sprache mit Funktionalität. So werden z.B. die JavaScript-Objekte wie Array oder Hash erweitert und zusätzlich neue Objekte wie Enumeration oder AJAX eingeführt.

4.3.2 EXT

Version: 1.0

Homepage: http://www.extjs.com/

Zweck: EXT (ehemals YUI-EXT) stellt eine JavaScript Bibliothek für grafische Benutzeroberflächen für Webseiten zur Verfügung. Es realisiert unter anderem Layouts mit Toolbars und Menus, Dialogfenster und verschiedene andere Grundelemente von Benutzerschnittstellen. Durch die Verwendung solch einer Bibliothek kann eine Web-Anwendung wie eine Desktop-Anwendung implementiert und umgesetzt werden (siehe Anforderung 2.1.8 und 2.1.16).

Bemerkung: Bei der eingebundenen EXT-Bibliothek musste die Implementierung angepasst werden, da die Version 1.0 von EXT nicht XHTML-konform ist. Somit würde ein Austausch auf eine andere Version möglicherweise Fehler nach sich ziehen. Es sollte dadurch die Version aus dem SVN (siehe 4.4) oder der CD benutzt werden.

4.3.3 Path-Parser

Version: 1.0

Homepage: http://kevlindev.com/dom/path_parser/index.htm

Zweck: Der Path-Parser bietet, wie der Name schon sagt, die Möglichkeit an ein SVG-Path zu parsen, um dann die jeweiligen Werte des Pfades auslesen zu können.

4.3.4 Log4JS

Version: 0.2

Homepage: http://log4js.sourceforge.net/

Zweck: Simpler Logger für das Erstellen von Log-Nachrichten. Folgende Nachrichtentypen sind damit möglich: DEBUG, ERROR, FATAL, INFO und WARN.

4.4 Projektstruktur

Das Projekt Oryx ist auf dem Subversion (SVN) Server [1] des Fachbereiches Business Process Technologie (BPT) verfügbar. Unter dem Tag LATEST_STABLE liegt die aktuell stabile Version des Editors. Mit dem Benutzer 'anonymous' und dem gleichwertigem Passwort ist ein Readonly-Zugriff möglich. Des Weiteren wurde ein Trac-System [2] für die Projektverwaltung und eine Google-Group [3] für die Projektkommunikation eingerichtet.
Auf der beigefügten CD ist eine Version des Editor vom 30. Juni 2007, inklusive dieser Bachelorarbeit, enthalten.

4.4.1 Ordnerstruktur

Das Projekt liegt in der folgenden Ordner-Struktur vor:

```
<SVN-ROOT>
 |- branches   // Aktuelle Branches des Editors
 |- tags       // Getaggte Versionen des Editors, z.B. LATEST_STABLE
 |- trunk
   |- doc         // Dokumentationen, Präsentationen, Bachelorarbeiten
   |- evaluation  // Einzelne Prototypen, Technologie Evaluation
   |- resources   // Bilder, Logos
   |- oryx        // Komplette Editor-Implementierung
      |- data        // Verschieden Daten, wie z.B. Stencil-Sets
      |- etc
      |- lib         // Externe Bibliotheken, wie z.B. Prototype, EXT
      |- shared      // Gemeinsame genutzte Daten,
      |              // wie z.B. DataManager, ERDF-Parser
      |- spec        // Spezifications-Tests für den Editor
      |- src         // Source-Code des Editors
         |- css         // CSS-Dateien
         |- images      // Bilder die benötigt werden
         |- script      // JavaScript-Dateien
            |- Core        // Core-Klassen
               |- Controls    // Controls, wie z.B. Docker und Magnet
               |- Math        // Klasse für mathematische Berechnungen
               |- StencilsSet // Klassen für das Stencil Set
               |- SVG         // Klassen für SVG-Repräsentation
            |- Plugins     // Alle Plugins
```

4.4.2 Seitenstruktur

Für Testzwecke liegen einige XHTML-Seiten mit verschiedenen Stencil-Sets im Ordner 'oryx' vor. Diese können auch lokal gestartet werden, wobei allerdings keine Datenspeicherung statt findet.

Nachstehend wird ein minimaler Seitenaufbau beschrieben, um denn Editor in einer XHTML-Seite anzuzeigen. Damit der Editor einwandfrei funktioniert, muss auf XML-Konformität geachtet werden. So müssen z.B. alle verwendeten XML-Namespaces und die aktuelle XML-Version angegeben werden.

[1] Link: https://svn.bpt.hpi.uni-potsdam.de/svn/b3mn/
[2] Link: https://svn.hpi.uni-potsdam.de/trac/b3mn/wiki/B7
[3] Link: http://groups.google.de/group/b3mn

```
<?xml version="1.0" encoding="utf-8"?>
<html   xmlns="http://www.w3.org/1999/xhtml"
        xmlns:b3mn="http://b3mn.org/2007/b3mn"
        xmlns:ext="http://www.extjs.com/ext">
        ...
```

Im Head-Tag der Seite muss ein „Profile“ mit der URL zum eRDF-Profile angegeben sein. Dadurch wird der eRDF-Parser aktiviert.

```
 <head profile="http://purl.org/NET/erdf/profile">
    ...
```

Innerhalb des Head-Tags müssen alle JavaScript-Dateien und andere Ressourcen referenziert werden die Oryx voraussetzt. Es ist auf die Reihenfolge zu achten, da es mögliche Abhängigkeiten in den Scripten gibt. Im Allgemeinen lädt der Editor alle spezifischen Ressourcen dynamisch nach. Was allerdings angegeben werden muss, sind alle verwendeten externen Bibliotheken.
So sollte als Erstes die Prototype-Bibliothek und der Logger eingebunden werden.

```
<script src="lib/prototype-1.5.1_rc3.js" type="text/javascript" />
<script src="lib/log4js.js" .../>
```

Als Zweites sollten die Dateien für die Bibliotheken Path-Parser und EXT hinzugefügt werden.

```
<script src="lib/path_parser.js" .../>

<script src="lib/ext-1.0/adapter/yui/yui-utilities.js" .../>
<script src="lib/ext-1.0/adapter/yui/ext-yui-adapter.js" .../>
<script src="lib/ext-1.0/ext-all-debug.js" ... />
<script src="lib/ext-1.0/ColorField.js" type="text/javascript" />
<style>
       @import url("lib/ext-1.0/resources/css/ext-all.css");
       @import url("lib/ext-1.0/resources/css/ytheme-aero.css");
</style>
```

Danach sollten die Bibliotheken hinzugefügt werden, die zwar vom Projekt Oryx stammen, aber nicht nur für den Editor vorhanden sind, sondern für alle Applikationen auf der Seite. So realisiert die Kickstart das Starten und Hinzufügen von JavaScript-Dateien während des Ladens der Seite. Der DataManager implementieren die Zwischenschicht, die in der Architektur (siehe 4.2) beschrieben wurde. Er nutzt den eRDF-Parser, der für das Parsen der Daten auf der aktuellen Seite zuständig ist.

```
<script src="shared/kickstart.js"   .../>
<script src="shared/erdfparser.js"  ... />
<script src="shared/datamanager.js" ... />
```

Nachdem müssen nun die Dateien angegeben werden die Editor-spezifisch sind und nur von diesem genutzt werden.

```
<script src="src/scripts/oryx.js" .../>
<style>
       @import url("src/css/theme_norm.css");
</style>
```

Zum Schluss müssen alle Ressourcen der eRDF-Prefixe angeben werden. In unserem Fall ist dies nur die von Oryx, da wir später nur diese verwenden werden. Diese sind wichtig, damit der eRDF-Parser feststellen kann, ob Einträge im DOM relevant für ihn sind oder nicht.

```
<link rel="schema.oryx" href="http://oryx-editor.org/" />
```

Um den Editor auf der Seite anzuzeigen, muss ein HTML-DIV-Element im Body-Tag der Seite eingefügt werden. Wichtig bei diesem Element ist, dass der Class-Name '-oryx-canvas' angegeben ist und dass ihm eine eindeutige Id zugewiesen wird. Mittels des eRDF-Parsers kann damit auf dieses Element zugegriffen werden und der Editor kann dynamisch alle verwendeten Objekte hinzufügen. Höhe und Breite können ebenfalls angegeben werden (initial wäre 1200px breit und 600px hoch). Für jedes dieser DIV-Elemente wird jeweils ein Editor generiert. D.h. es wäre auch möglich mehrere Editoren auf einer Seite anzuzeigen.

```
<div class="-oryx-canvas" id="oryx-canvas123"> ... </div>
```

In diesem DIV-Element können noch verschiedene Einstellungen angegeben werden. So kann z.B. das Stencil Set angegeben werden, welches initial geladen werden soll. Es kann aber auch angegeben werden, ob der Editor im Readonly-Modus oder im Vollbild-Modus geladen werden soll oder nicht.

```
<a rel="oryx-stencilset" href="./data/stencilsets/bpmn/bpmn.json"></a>
<span class="oryx-mode">readonly</span>
<span class="oryx-mode">fullscreen</span>
```

Des Weiteren wird in diesem DIV des Editors auf alle Daten in der aktuellen Seite referenziert, die gerendert werden sollen.

```
<div class="-oryx-canvas" id="oryx-canvas123">
      ...
      <a rel="oryx-render" href="#1"></a>
      ...
</div>
...
<div id="1">
      <span class="oryx-type">http://b3mn.org/stencilset/bpmn#Task</span>
</div>
```

Mehr Information darüber, was und wie gespeichert wird, entnehmen Sie bitte der Bachelorarbeit „Embedding Business Process Data into the Web“ von Martin Czuchra [4]. In dieser ist eine genaue Beschreibung des DataManagers und des eRDF-Parsers dargestellt.

4.5 Implementierung

4.5.1 Klassenkonzept

Da JavaScript prototypisch ist, besitzt es grundlegend erstmal kein Klassenkonzept. Das implementierte Konzept basiert auf dem von Joshua Gertzen [7] vorgestellten Klassenkonzept. Es beschreibt, wie eine Basis-Klasse implementiert sein muss, damit vererbte Klassen nicht die gleichen Methoden überschreiben, sondern diese im

selben Kontext gerufen werden können.
Die von uns benutzte Basis-Klasse ist die Klasse 'Clazz', die durch 'Clazz.extend(...)' vererbt werden kann.
Dadurch ist es z.B. möglich, dass 'ORYX.Core.AbstractShape' von 'ORYX.Core.UIObject' abgeleitet ist, dieser aber noch den Konstruktor der Basis-Klasse 'UIObject' rufen kann.

```
ORYX.Core.UIObject = Clazz.extend({
     construct: function() {
          ...
     }
})

ORYX.Core.AbstractShape = ORYX.Core.UIObject.extend({
     construct: function() {
           arguments.callee.$.construct.apply(this, arguments);
           ...
     }
})
```

4.5.2 Pakete

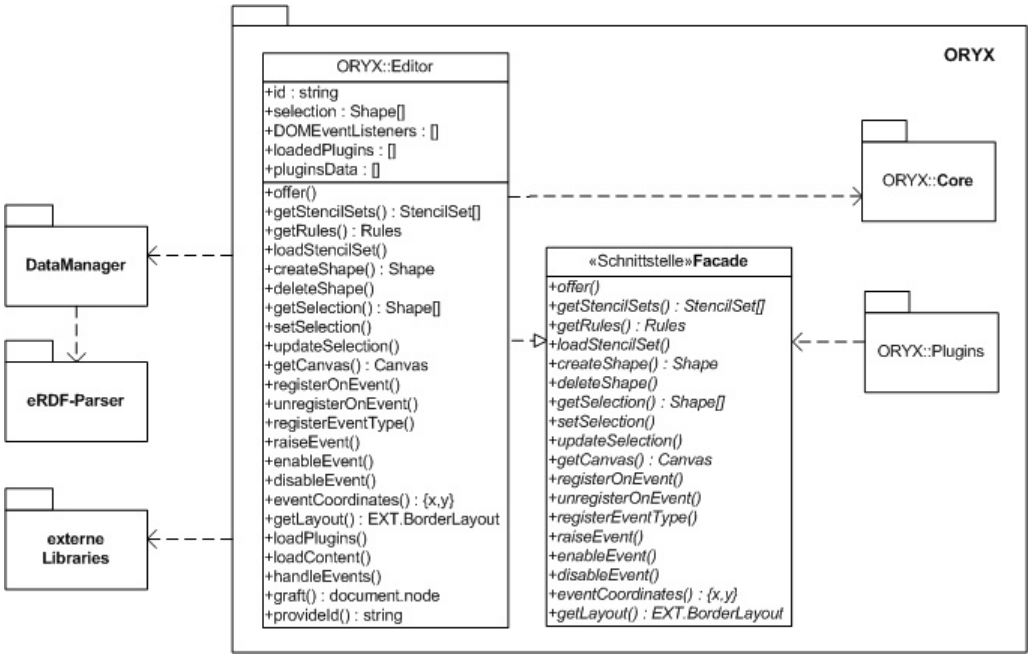

Abbildung 4.5: Paket - Diagramm

Die Oryx Packetstruktur ist in mehrere Teile gegliedert (siehe Abbildung 4.5). So gibt es die Zwischenschicht (DataManager und eRDF-Parser, siehe Architektur 4.2) und die externen Bibliotheken, wie z.B. Prototype und EXT (siehe 4.3). Daneben gibt es das Paket Oryx. Es beinhaltet die Pakete Core und Plugins. Beide implementieren die in der Architektur (siehe 4.2) beschriebene Funktionalität. Des Weiteren gibt es den Editor, der für die Implementierung der Kernfunktionalität zuständig ist.

Es lädt die Objekte, die in der aktuellen Ressource verfügbar sind und setzt den Plugin-Mechanismus um. Er initialisiert also die Plugins und stellt eine API für sie bereit.

Folgend werden noch einzelne Packete beschrieben, die Teil des Core Pakets sind.

ORYX.Core.StencilSet

Das Stencil Set Paket bietet die Implementierung der Stencil Set Spezifikation an. So können neue Stencil Sets und neue Stencils geladen werden und die Grammatik der geladenen Stencil Sets kann durch verschieden Objekte abgefragt werden. Ausführliche Information über die Stencil Set Spezifikation wird in der Bachelorarbeit von Nicolas Peters [11] beschrieben.

ORYX.Core.SVG

Das Paket SVG kapselt den Zugriff und die Manipulation von einzelnen SVG Elementen. So wird z.B. ein Pfad-Parser oder die Verwendung von Markers in SVG angeboten. Ausserdem werden Textfelder für den Editor bereitgestellt.

ORYX.Core.SVG-Drag

SVG-Drag bietet die Möglichkeit SVG-Elemente mittels Drag und Drop zu verschieben. Es ist als Funktionsaufruf realisiert, welcher nur das Mouse-Down-Event und das aktuelle Objekt aus dem Editor erwartet.

ORYX.Core.Math

Das Paket Math bietet mathematische Methoden, wie z.B. Schnittpunktberechnungen an. So ist dort folgende Berechnung implementiert: Schnittpunkt zwischen einem Punkt und einer Linie, zwischen einem Punkt und einem Kreis und zwischen einem Punkt und einem Polygon.

4.5.3 Klassen

Folgend werden alle ORYX.Core-Klassen beschrieben. Die ORYX.Core-Klassenhierarchie ist im Diagramm 4.6 dargestellt.

ORYX.Core.Bounds

Die Bounds realisiert die Verarbeitung und Berechnung von einem, mittels zwei Punkten aufgespannten, Rechteck, welches die Dimension eines Shapes angibt. Neben den Getter-Methoden (upperLeft, lowerRight, width, height) bietet es auch Setter-Methoden, die entweder absolut oder relativ arbeiten (z.B. set, moveBy, moveTo, ...).

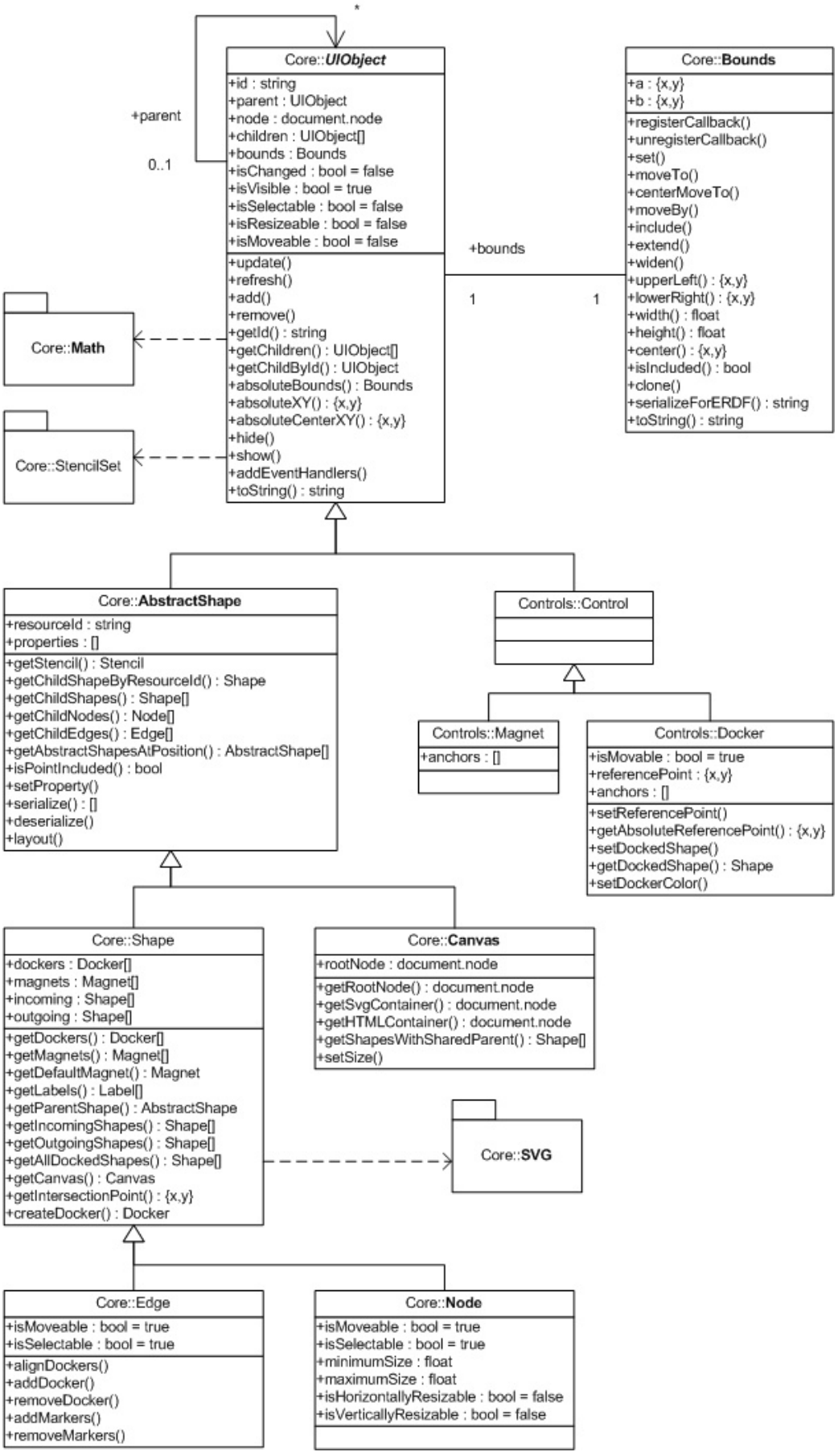

Abbildung 4.6: Klassen - Diagramm - ORYX.Core

ORYX.Core.UIObject

UIObject ist die Basis-Klasse aller im Editor vorkommenden graphischen Elemente. Zu dieser zählen neben den eigentlichen Shapes auch die Dockers und Magnets oder die Canvas. Es bietet grundlegende Funktionalitäten und erste Attribute, wie z.B. isResizable, isSelectable, isMoveable an, die angeben, ob ein Element die Größe ändern darf, selektierbar oder bewegbar ist. Diese Klasse implementiert die Add und Remove-Methoden von Objekten, worüber ein UIObject wieder ein UIObject hinzufügen oder löschen kann. Es werden Methoden bereitgestellt, um an Kind-Elemente bzw. über eine Id an ein Kind-Element zu gelangen. Des Weiteren bietet es die Möglichkeit an die absolute Bounds bzw. absolute Position zu gelangen. Diese werden anhand der Bounds der Parents berechnet. Ferner wird hier das Ein- und Ausblenden eines UIObjects realisiert. Zusätzlich wird der Refresh-Algorithmus in dieser Klasse implementiert. D.h wenn die Bounds geändert wurde und die 'Update'-Methode gerufen wird, wird beim eigenen Objekt die 'refresh'-Methode gerufen und bei den Kind-Elementen die 'Update'-Methode. Die 'refresh'-Methode ist hier noch abstrakt gehalten und muss in den abgeleiteten Klassen realisiert werden. Dort muss beschrieben werden, wie ein Element sich anhand der Bounds neu ausrichtet. Zu guter Letzt wird hier der Mouse-Event zyklus realisiert. D.h. jedes UIObject ist für seine Mouse-Events selber verantwortlich. Die Klasse UIObject realisiert lediglich, dass ein Callback bei der Initialisierung übergeben werden kann, der bei Mouse-Events gerufen wird.

ORYX.Core.AbstractShape

AbstractShape kapselt die Methoden für die Canvas und die Shapes. Es werden hauptsächlich Methoden bereitgestellt, um an Kind-Elemente zu gelangen. So ist es unter anderem möglich an Kind-Elemente mittels der ResourceId, also der Id aus dem DOM der aktuellen Seite, zu gelangen. Es ist ausserdem möglich an alle Kind-Elemente, die Knoten und/oder Kante sind, zu kommen. Auch durch eine XY-Koordinate besteht die Möglichkeit auf Objekte im Editor zu referenzieren. Dabei ist das letzte Objekt in der Liste das oberste Objekt im Editor.

ORYX.Core.Canvas

ORYX.Core.Canvas realisiert die eigentliche Zeichenfläche und ist zusätzlich das Root-Element für alle Shapes, also das Eltern-Element aller Shapes. Dadurch kommt man durch die Canvas an alle Shapes, die im Editor verfügbar sind. Zusätzlich hält es Referenzen auf einen HTML bzw. SVG Container, der von den Plugins genutzt werden kann, um Elemente dem DOM hinzuzufügen.
Es wird die oberste SVG-Struktur realisiert.

```
<svg>
  <g>
      <g class="stencils">
          <g class="me">...</g>
          <g class="children">...</g>
          <g class="edge">...</g>
      </g>
      <g class="svgcontainer">...</g>
  </g>
</svg>
```

In 'me' werden für die Zeichenfläche relevante Elemente abgelegt. Im G-Element 'children' werden alle Knoten abgelegt, die hierarchisch gegliedert werden (siehe ORYX.Core.Shape). Da Kanten keine Hierarchie aufbauen können, werden sie im G-Element 'edge' abgelegt. Dadurch wird eine Kante immer als oberstes Element im Editor angezeigt.
Ferner ist realisiert, dass alle Properties aus dem Stencil Set ausgelesen werden und auf diesen entweder lesend oder schreibend Bezug genommen werden kann. Ausserdem bietet es noch die Möglichkeit Elemente zu serialisieren und zu deserialisieren. D.h., dass ein Element in eine allgemein gültige Form gebracht wird und anhand dieser Daten auch wieder reproduziert werden kann. Bei der Serialisierung wird ein Array zurückgegeben, in dem Wert-Paare und Meta-Informationen enthalten sind. So ist der Name der Eigenschaft und der Wert angegeben. Zusätzlich wird der Type ('resource' oder 'literal') und der Prefix für die Eigenschaft angegeben, die für den DataManager wichtig sind. Bei der Deserialisierung müssen genau diese Daten wieder übergeben werden.

```
[{prefix, type, name, value},...] serialize()
void deserialize([{prefix, type, name, value},...])
```

Erste Ansätze eines Layouting-Algorithmus ist hier implementiert. So kann im Stencil Set, mittels eines Callbacks, angegeben werden, wie ein Elemente sich an anderen Elementen ausrichtet. Dieses wird dann beim Update überprüft. Einen genaueren Einblick in (de)serialize und dem Layouting liefert Nicolas Peters in seine Bachelorarbeit [11].

ORYX.Core.Shape

Shape kapselt Methoden die nur für Shapes, also Kanten und Knoten, zuständig sind. Es realisiert die SVG-DOM Struktur für ein Shape:

```
...
<g>
    <g class="stencils">
        <g class="me">...</g>
        <g class="children">...</g>
    </g>
    <g class="controls">
        <g class="dockers">...</g>
        <g class="magnets">...</g>
    </g>
</g>
...
```

So werden die eigene SVG-Repräsentation in das G-Element mit dem Class-Name 'me' abgelegt. Kind-Elemente werden bei 'children', Docker bei 'dockers' und Magneten bei 'magnets' abgelegt. So kann sichergestellt werden, dass Kind-Elemente immer über dem eigenen Element angezeigt werden und das Docker und Magnets immer oben auf liegen. Allerdings können nur bei Knoten neue Knoten, als Kind-Elemente, hinzugefügt werden. Kanten hingegen werden auf oberster Ebene abgelegt (siehe ORYX.Core.Canvas).
Es hält noch die Objekt-Referenzen auf alle Dockers und Magnets und auf alle eingehenden und ausgehenden Shapes. Ausserdem kann bei einem Shape nachgefragt werden, wie der Schnittpunkt mit dem Element und einer Linie ist, wenn der eine Punkt innerhalb und der andere Punkt ausserhalb des Elements liegt. Dies wird für die korrekte Kantendarstellung benötigt.

ORYX.Core.Node

Node ist die konkrete Klasse eines Knotens im Editor. Ein Knoten wird durch das übergebene Stencil beschrieben und damit erzeugt. So ist im Stencil zum einen die SVG Repräsentation angegeben und zum anderen verschiedene Einstellungen die für den Knoten wichtig sind. Es ist z.B. angegeben wie das Element die Größe ändern darf oder wie viele Magneten das Element besitzt. Falls keine Magneten angegeben sind, wird Initial eins in der Mitte platziert.

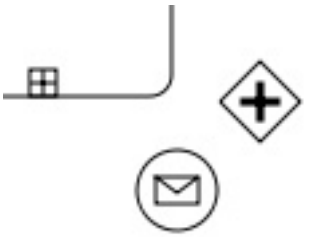

Abbildung 4.7: Knoten

ORYX.Core.Edge

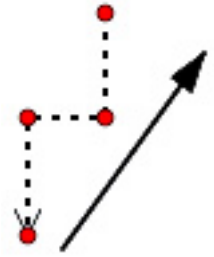

Abbildung 4.8: Kante

Edge ist die konkrete Klasse einer Kante im Editor. Auch hier wird die Kante durch das übergebene Stencil beschrieben und daraus erzeugt. In der SVG-Repräsentation können Pfade angegeben werden, die die Form der Kante beschreiben. Es wird bei der Erstellung einer Kante an jedem Ende eines Pfade ein Docker hinzugefügt. Anhand dieser können die Referenzpunkte einer Kante verändert werden. Nach der Initialisierung können beliebig viele Docker einer Kante hinzugefügt werden.

ORYX.Core.Controls.Docker

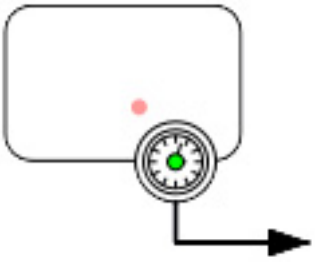

Abbildung 4.9: Docker

Ein Docker ist immer einem Eltern-Shape zugewiesen. Beide beeinflussen sich bei der Positionierung gegenseitig. So richtet sich ein Shape am Docker aus und umgekehrt. Zusätzlich kann sich ein Docker an einem anderen Shape andocken. Dabei positioniert sich der Docker, wenn er einer Kante zugewiesen ist, am Schnittpunkt der Kante mit dem angedocktem Shape. Oder, wenn er einem Knoten zugewiesen wurde, an der Verlängerung des Zentrums zum Rand. In dieser Klasse kann somit das angedockte Shape gesetzt oder abgefragt werde. Zusätzlich kann noch der Referenzpunkt des Dockers neu gesetzt werden.

ORYX.Core.Controls.Magnet

Ein Magnet dient als Snap-Punkt für einen Docker. Somit können explizite Punkte angegeben werden, die durch einen Docker getroffen werden sollen.

ORYX.CONFIG

Diese Klasse stellt verschiedene Konstanten zur Verfügung, die global genutzt werden können. So werden unter anderem folgende Werte bereit gestellt:

```
ORYX.CONFIG = {
    ...
    /* Plugins */
    PLUGINS_CONFIG:         "scripts/Plugins/plugins.xml",
    PLUGINS_FOLDER:         "scripts/Plugins/",
```

```
    /* Namespaces */
    NAMESPACE_ORYX:        "http://www.b3mn.org/oryx",
    NAMESPACE_SVG:         "http://www.w3.org/2000/svg",

    /* UI */
    CANVAS_WIDTH:          2970, /* Din A4 - Size*/
    CANVAS_HEIGHT:         2100,
    SELECTED_AREA_PADDING: 4,
    GRID_DISTANCE:         30,
    GRID_ENABLED:          true,
    ZOOM_OFFSET:           0.1,

    /* Shape */
    MINIMUM_SIZE:          20,
    MAXIMUM_SIZE:          10000,

    /* Property type names */
    TYPE_STRING:           "string",
    TYPE_BOOLEAN:          "boolean",
    TYPE_INTEGER:          "integer",
    TYPE_FLOAT:            "float",
    TYPE_COLOR:            "color",
    TYPE_DATE:             "date",
    TYPE_CHOICE:           "choice",
    TYPE_URL:              "url",
    TYPE_RICHTEXT:         "richtext",

    /* Selection Shapes Highlights */
    SELECTION_HIGHLIGHT_SIZE:  5,
    SELECTION_HIGHLIGHT_COLOR: "#4444FF",

    SELECTION_VALID_COLOR:     "#00FF00",
    SELECTION_INVALID_COLOR:   "#FF0000",

    ...
};
```

5. Plugins

In diesem Kapitel wird das Plugin Konzept genauer beschrieben und bereits vorhandene Plugins aufgelistet. Um dies anschaulicher darzustellen, werden zwei Anleitungen für die Erstellung eines Plugins genutzt.

5.1 Das Plugin Konzept

Um ein Plugin zu realisieren müssen zwei Dinge beschrieben werden. Zum einen muss das Plugin geschrieben und zum zweiten muss dieses in der 'plugins.xml' eingetragen werden. Bei einem Plugin stellt sich die Frage, ob es Funktionalität anbietet oder ob es vorhandene Funktionalität bereitstellt (z.B. mit durch eine GUI). Beides kann auch gemeinsam implementiert werden.

Alle angegebenen Plugins werden bei der Initialisierung des Editors instantiiert und sind somit verfügbar. Dabei erhalten sie einerseits ein Interface vom Editor und zum anderen werden die relevanten Konfigurationsdaten übergeben, welche in der 'plugins.xml' als 'Property' angegeben sind. Wenn sie Funktionalität am Editor registrieren wollen, geschied dies über die offer-Methode aus dem Interface (siehe hierzu 5.1.3). Möchte ein Plugin auf die Daten der anderen Plugins Bezug nehmen, um sie z.B. als grafische Benutzerschnittstelle anzubieten, muss die Methode 'registryChanged' im Plugin implementiert werden (siehe 5.1.4).

5.1.1 Schnittstelle zum Editor

Jedem Plugin wird eine Schnittstelle zum Editor bereitgestellt. Im folgenden wird die Schnittstelle (siehe Abbildung 5.1), die den Plugins zur Verfügung steht, beschrieben.

Abbildung 5.1: Facade

Die Funktionalität, die angeboten werden soll, muss am Editor registriert werden. Hierbei müssen Meta-Daten angegeben werden, die im Abschnitt 5.1.3 beschrieben sind.

```
void offer(Object)
```

Es gibt die Möglichkeit an Information der geladenen Stencil Sets zu gelangen, so kommt man an die Objekte der Stencil Sets oder an das Objekt der Rules-Klasse. Des Weiteren wird das Neuladen eines Stencil Sets angeboten; hierbei muss der Pfad übergeben werden.

```
ORYX.Core.StencilSet.StencilSet[] getStencilSets()
ORYX.Core.StencilSet.Rules getRules()
void loadStencilSet(string)
```

Es werden Methoden angeboten, um neue Shapes zu erzeugen oder vorhandene Shapes zu löschen. Beim Löschen eines Shapes muss lediglich das zu löschende Shape übergeben werden. Um ein neues Shape zu erstellen, kann ein Objekt mit folgende Attributen übergeben werden:

```
namespace: string
    Namensraum des Stencil Sets.

type: string
    Typ des Stencils, welches im Namensraum beschrieben wurde.

position: {x,y} = {x:100, y:200}
    Relative Center-Position des neuen Shapes.

parent: ORYX.Core.AbstractShape = ORYX.Core.Canvas
    Eltern Shape des neuen Shapes.

connectedShape: ORYX.Core.Shape = undefined
    Vorheriges Shape, welches verbunden werden soll.

connectingType: string = undefined
    Kanten-Type mit dem verbunden werden soll. Nur in Verbindung
    mit 'connectedShape'.
```

oder

```
serialize: {prefix, type, name, value}[]
    Serialisierung eines Shapes. Anhand dieser wird ein Shape
    neu erzeugt und deserialisiert.
```

Ein Shape kann somit entweder mit dessen Eigenschaften oder mit dem Serialisierungs-Objekt eines Shapes erzeugt werden.

```
ORYX.Core.Shape createShape(Object)
void deleteShape(ORYX.Core.Shape)
```

Des Weiteren gibt es Methoden, um auf die aktuelle Selektion Bezug nehmen zu können. Sie kann entweder ausgelesen, neu gesetzt oder aktualisiert werden.

```
ORYX.Core.Shape[] getSelection()
void setSelection(ORYX.Core.Shape[])
void updateSelection()
```

Um Bezug auf die Events zu nehmen, gibt es eine Reihe von Methoden. So kann eine Funktion an ein Event registriert werden, neue Events können realisiert und vorhandene Events können geworfen werden. Dabei ist allerdings zu beachten, dass beim Event mindestens das Attribut 'type' gesetzt ist, welches den Typ des Events angibt.
Das Namensschema für den Eventtyp sollte wie folgt lauten:
'<Plugin-Name>.<Event-Name>'.
Zusätzlich existiert die Möglichkeit Events zu deaktivieren bzw. zu aktivieren. Alle Events die im Editor vorkommen, sind im Anhang A.1 beschrieben.

```
void registerOnEvent(string, function)
void unregisterOnEvent(string, function)
void registerEventType(string)
void raiseEvent(window.event || Oryx.Event)
void enableEvent(string)
void disableEvent(string)
```

Ferner gibt es noch eine Methode, die eine Mouse-Position anhand der SVG-Matrix korrigiert, oder welche, die die Canvas oder das Layout-Objekt des Editors zurückgeben.

```
{x,y} eventCoordinates(Mouse.Event)
Ext.BorderLayout getLayout()
ORYX.Core.Canvas getCanvas()
```

5.1.2 Plugin Konfigurationsdaten

Als zweites Parameter erhält jedes Plugin Daten, die aus der 'plugins.xml' stammen. Sie beinhaltet den Namen der Klasse des Plugins und die Pfadangabe zur JavaScript-Datei. Ausserdem werden Eigenschaften mit übergeben, die in der 'plugins.xml' angegeben sind. Sie sind entweder im Knoten des jeweiligen Plugins oder im 'Properties'-Knoten der Datei beschrieben. So sollte, wenn die Eigenschaft für alle Plugins zugänglich gemacht werden soll, diese in den 'Properties'-Knoten am Ende der 'plugins.xml' eingetragen werden. Soll die Eigenschaft nur für das jeweilige Plugin gelten, reicht es aus, wenn dies im Knoten des Plugins beschrieben wird.

```
{
   name:       string,
   source:     string,
   properties: [{<Attributname>:<Attributwert>, ...}, ...]
}
```

5.1.3 Funktionalität anbieten

Plugins die Funktionalität anbieten, müssen dies durch die 'offer'-Methode im Interface des Editors realisieren. Dabei muss ein Objekt mit folgenden Attributen übergeben werden, welches danach von den anderen Plugins interpretiert werden kann.

```
name: string
    Name der Funktionalität

functionality: function()
    Callback der Funktionalität

group: string
    Gruppen-Name

icon: string
    Grafische Repräsentation (Absolute, oder
    Relative zum Root-Ordner)

description: string
    Beschreibung der Funktionalität

index: int
    Gibt an, in welcher Priorität diese Funktionalität
    gegenüber anderen, gleicher Gruppe, steht
    (je kleiner, desto wichtiger). Dadurch kann eine
    Sortierung der Funktionalität gestaltet werden.

minShape: int
    Mindestanzahl an Shapes, ab wann die Funktionalität
    aktiviert ist

maxShape: int
    Maximale Anzahl an Shapes, bis wann die Funktionalität
    aktiviert ist

isEnabled: bool function()
    Es kann ein Callback angeben werden, welcher gerufen
    wird, um zu sehen ob diese Funktionalität aktiviert
    ist. Je nachdem muss sie 'true' oder 'false' zurückgeben
```

5.1.4 Vorhandene Funktionalität bereitstellen

Plugins, die die vorhandene Funktionalität anzeigen sollen, müssen, um an die vorhandenen Daten zu gelangen, folgende Methode implementieren:

```
registryChanged: function(Object){...}
```

Die Methode wird aufgerufen sobald alle Plugins verfügbar sind. Als Argument bekommt sie ein Array mit den Daten übergeben, die die Plugins bereitgestellt haben (siehe 5.1.3). Diese Daten können dann genutzt werden, um die gewünschte Information zu extrahieren und dem User bereit zu stellen.

5.2 Plugin - Tutorial

Als Beispiel für die Erstellung von Plugins werden hier zwei Plugins beschrieben: ein Plugin, welches einen Docker zu einer Kante hinzufügt und ein Plugin, was für das Verschieben eines Dockers zuständig ist. Die beiden Beispiele sind so gewählt, dass möglichst viel vom Interface benutzt wird. Des Weiteren sind sie bereits implementiert und können im Plugin-Ordner unter 'addDocker.js' und 'dragDocker.js' eingesehen werden. Beide Plugins werden getrennt voneinander beschrieben, um eine klare Vorstellung zu erhalten, welche Aufrufe zu welchem Plugin gehören.

5.2.1 Tutorial 1: 'AddDocker'

Das Plugin AddDocker soll bei einem Klick auf eine Kante einen Docker hinzufügen, dafür muss es allerdings vorher aktiviert worden sein.

Zuerst muss eine neue Datei angelegt werden; in diesem Fall heißt sie 'addDocker.js'. Um eine neue Klasse zu schreiben, verwenden wir das Klassenkonzept, welches bereits im Editor Anwendung gefunden hat. Wir erben somit von 'Clazz' und erstellen erstmal einen Konstruktor.

```
var AddDocker = Clazz.extend({
    construct: function(facade, pluginsdaten) {
    }
});
```

Im Konstruktor werden zwei Argumente übergeben. Einmal die Facade für den Editor und zum andern die Plugin-Daten, die in der 'plugins.xml' angegeben sind. Für dieses Plugin ist nur die Facade relevant. Sie wird somit im Konstruktor als globale Variable abgespeichert.

```
        this.facade = facade;
```

Als nächstes brauchen wir eine Variable die speichert, ob die Funktionalität aktiviert wurde. Zusätzlich soll auf das Mouse-Down Event reagiert werden das vom Editor kommt, um an dieser Stelle möglicherweise einen Docker hinzufügen zu können. Die hier registrierte Methode wird später genauer erläutert.

```
        this.isEnabled = false;
        this.facade.registerOnEvent('mousedown',this.handleMouseDown.bind(this));
```

Was nun noch in den Konstruktor kommt, ist das Registrieren der Funktionalität beim Editor. Es soll lediglich die Variable 'isEnabled' auf 'true' gesetzt werden, wenn die Funktionalität aktiviert wird. Was noch angegeben werden, muss ist der Name der Funktionalität und ein Gruppe, damit später die Funktionalität gruppiert werden kann. Ausserdem sollte angegeben sein, ab wann die Funktionalität verfügbar ist. Entweder durch 'minShape' und 'maxShape', also minimale und maximale Anzahl des selektierten Shapes oder durch ein Callback der 'true' oder 'false' zurück gibt. Wir geben als 'min-' und 'maxShapes' '0' an, so dass die Funktionalität immer verfügbar ist.

```
        this.facade.offer({
             'name':"Add Docker",
             'functionality': function(){this.isEnabled = true}.bind(this),
             'group': "Docker",
             'icon': ORYX.PATH + "images/vector_add.png",
             'description': "Add a Docker to an edge",
             'index': 1,
             'minShape': 0,
             'maxShape': 0});
```

Damit beenden wir die Arbeit am Konstruktor und müssen nun die Methode 'handleMouseDown' implementieren. Da die Methode durch ein Mouse-Event aufgerufen wird, bekommen wir einmal das Event und zum anderen das aktuelle Shape, also das Shape auf das geklickt wurde.

```
handleMouseDown: function(event, shape) {
```

Es muss überprüft werden, ob die Funktionalität aktiviert wurde und ob das übergebene Shape eine Kante ist. Wenn dem nicht so ist, wird die Methode 'handleMouseDown' abgebrochen.

```
    if(!this.isEnabled || !shape instanceof ORYX.Core.Edge) { return };
```

Ist die Funktionalität aktiviert und das aktuelle Shape ist eine Kante, können wir den Docker an der aktuellen Mouse-Position hinzufügen. Im Vorfeld sollte die Position an der aktuellen SVG-Matrix korrigiert werden.

```
    var evPos = this.facade.eventCoordinates(event);
    shape.addDocker(evPos);
}
```

Im Anschluss wird das Plugin in die 'plugins.xml' eingetragen. Dabei wird zum einen der Pfad der JavaScript-Datei angegeben und zum anderen der Namen der Klasse, die instantiiert werden soll.

```
<?xml version="1.0" encoding="utf-8"?>
<config>
    <plugins>
        ...
        <plugin source="addDocker.js" name="AddDocker" />
        ...
```

5.2.2 Tutorial 2: 'DragDocker'

Für dieses Plugin erstellen wir erneut eine JavaScript-Datei: 'dragDocker.js'. In dieser wird nun die Klasse 'DragDocker' erstellt. Sie soll realisieren, dass ein Docker und das dazugehörige Shape gedraggt werden kann. Zusätzlich soll es überprüfen, ob der Docker sich an das darunter liegende Shape andocken kann. Wenn das der Fall ist, wird beim Mouse-Up Event dieser Docker an das Shape angedockt. Um dies für den Benutzer zu signalisieren, ob er andocken darf oder nicht, wird das 'Highlighting-Plugin' benutzt, um das Shape farblich zu markieren. Die Farbe für das Highlighting wird als ein Property in der 'plugins.xml' stehen.

Zuerst wird das Plugin in die 'plugins.xml' eingetragen. Zusätzlich zu Name und Pfad des Plugins fügen wir eine Eigenschaft ein, welches die beiden Farben angibt. Da das Property nur für dieses Plugin relevant ist, wird es in das Plugin-Tag geschrieben. Sollte es für alle interessant sein, könnte man es in den Knoten 'Properties' schreiben, welches am Ende der 'plugins.xml' steht.

```
        ...
        <plugin source="dragDocker.js" name="DragDocker">
              <property name="color" valid="#00FF00" invalid="#FF0000" />
        </plugin>
        ...
```

Nun müssen die Einstellungen aus der 'plugins.xml' herausgelesen werden. Dies geschieht über das zweite Argument, welches bei der Initialisierung übergeben wird. Um den entsprechenden Eintrag zu finden, wird das Array durchsucht.

```
var DragDocker = Clazz.extend({
     construct: function(facade, plugindaten) {
          this.facade = facade;

          var colorProperty = plugindaten.properties.find(
                                function(prop){
                                       return prop.name && prop.name === 'color'
                                });
          this.VALIDCOLOR   = colorProperty.valid
          this.INVALIDCOLOR = colorProperty.invalid
```

Weiter möchten wir uns auf das Mouse-Down eines Elements registrieren, überprüfen zu können, ob es ein Docker ist.

```
          this.facade.registerOnEvent('mousedown', this.handleMouseDown.bind(this));
     },
```

Hier findet der Konstruktor seinen Abschluß. Als nächstes wird die Implementierung der Mouse-Down-Methode beschrieben. In dieser sollte zuerst geprüft werden, ob es sich bei dem aktuellen Element auch tatsächlich um einen Docker handelt.

```
     handleMouseDown: function(event, uiObj) {
          if(!uiObj instanceof ORYX.Core.Controls.Docker) {return}
```

Wenn dem so ist, wird dieser zwischengespeichert und von seinem angedocktem Shape gelöst. Danach aktivieren wir das Draggen des aktuellen Dockers. Diese Funktion erwartet das Mouse-Down Event und das aktuelle UIObject. Hierzu können zusätzlich noch Callbacks für Mouse-Move und Mouse-Up übergeben werden.

```
          this.docker = uiObj;
          this.docker.setDockedShape(undefined);

          var option = {
                         movedCallback: this.dockerMoved.bind(this),
                         upCallback: this.dockerMovedFinished.bind(this)
                       };
          ORYX.Core.UIEnableDrag(event, this.docker, option);
     },
```

Die Mouse-Down Methode ist nun abgeschlossen. Wir kommen nun zu den beiden übergebenen Callbacks, für Mouse-Move und Mouse-Up. In der Mouse-Move Methode soll nun überprüft werden, ob an das darunter liegende Shape angedockt werden darf. Wenn sich der Docker mit dem Shape verbinden kann, entsteht eine Verbindung zwischen den beiden Shapes.
Zuerst werden alle Shapes ermittelt, die an der aktuellen Mouse-Position liegen. Die Position muss auch wieder durch die SVG-Matrix korrigiert werden.

```
     dockerMoved: function(event) {
          var pos    = this.facade.eventCoordinates(event);
          var shapes = this.facade.getCanvas().getAbstractShapesAtPosition(pos);
```

Danach wird auf das oberste Shape referenziert, welches nicht das Shape des dazugehörigen Dockers ist.

```
var uiObj = shapes.pop();
uiObj = this.docker.parent === uiObj ? shapes.pop() : uiObj;
```

Ist dieses Shape die Zeichenfläche, wird das Highlighting, welches möglicherweise aktiv ist, mittels eines Event-Aufrufs ausgeblendet und die Variablen zurückgesetzt.

```
if(uiObj instanceof ORYX.Core.Canvas) {
      this.facade.raiseEvent({
                              type:'highlight.hideHighlight',
                              highlightId:'dragDocker'
                            });
      this.validShape = undefined;
} else {
```

Falls das Shape nicht die Zeichenfläche ist, wird nun beim Stencil Set nachgefragt, ob der Docker sich bei dem Shape andocken darf. Dafür muss ein Source-, Edge- und Target-Objekt angegeben werden. Diese geben an von welchem, über welches und zu welchem Shape die Verbindung geprüft werden soll.

Um die Objekte richtig zu setzen, sollte zunächst überprüft werden, ob das dazugehörige Shape zum Docker, einer Kante oder einem Knoten gehört. Gehört es zu einem Knoten, wird als Source-Objekt das darunter liegende Shape gesetzt. Als Target-Objekt und Edge-Objekt wird das zum Docker gehörige Shape übergeben. Das bedeutet, wenn ein Knoten sich an einen Knoten andocken möchte, ist zu überprüfen, ob der Knoten Folge-Shape des anderen Knotens sein darf.
Das Ergebnis wird dann zwischengespeichert.

```
if(this.docker.parent instanceof ORYX.Core.Node) {
       var isValid  = this.facade.getRules().canConnect({
                            sourceShape: uiObj,
                            edgeShape:   this.docker.parent,
                            targetShape: this.docker.parent
                          });
       this.validShape = isValid ? uiObj : undefined;
} else {
```

Handelt es sich dabei um eine Kante, muss geprüft werden, ob der erste oder aber der letzte Docker angedockt ist. Wichtig ist somit ob die Kante von einen Shape kommt oder zu einem Shape referenziert ist. Wenn dies nicht der Fall ist, wird ermittelt, ob es sich um den Anfangs- oder Enddocker handelt, um bestimmen zu können, ob das darunter liegende Shape entweder als Source- oder Target-Objekt fungiert. Davon ausgehend werden nun die Parameter gesetzt.

```
       var d = this.docker;
       var dPd = d.parent.dockers ;

       var source = dPd.first().getDockedShape();
       source = source ?
                    source :
                    (dPd.first() === d ? uiObj : undefined);

       var target = dPd.last().getDockedShape();
       target = target ?
```

```
                                target :
                                (dPd.last() === d ? uiObj : undefined);

                        var isValid  = this.facade.getRules().canConnect({
                                    sourceShape: source,
                                            edgeShape:   this.docker.parent,
                                            targetShape: target
                                       });
                        this.validShape = isValid ? uiObj : undefined;
                }
```

Abhängig von der Antwort vom Stencil Set wird nun das Highlighting gesetzt. Dafür wird mittels eines Events dem Highlight-Plugin mitgeteilt, dass es die angegebenen Shape farblich markieren soll.

```
                this.facade.raiseEvent({
                           type:        'highlight.showHighlight',
                           highlightId: 'dragDocker',
                           elements:    [uiObj],
                           color:       this.validShape ?
                                                this.VALIDCOLOR :
                                                this.INVALIDCOLOR
                     });
          }
     },
```

Die Mouse-Move Methode ist nun abgeschlossen. Abschließend wird die Methode für Mouse-Up beschrieben. Hier wird, falls die aktuelle Verbindung korrekt ist, der Docker an das angegebene Shape angedockt. Zusätzlich wird die Selektion aktualisiert.

```
     dockerMovedFinished: function(event) {
          if(this.validShape)){
                this.docker.setDockedShape(this.validShape);

                this.facade.updateSelection();
      }
     }
}
```

5.3 Vorhandene Plugins

Nachstehend werden alle Plugins genannt, die derzeit umgesetzt sind. Dabei sind unter anderem die kompletten Funktionalitäten implementiert, die in der Anforderung 2.1.9 angegeben sind.

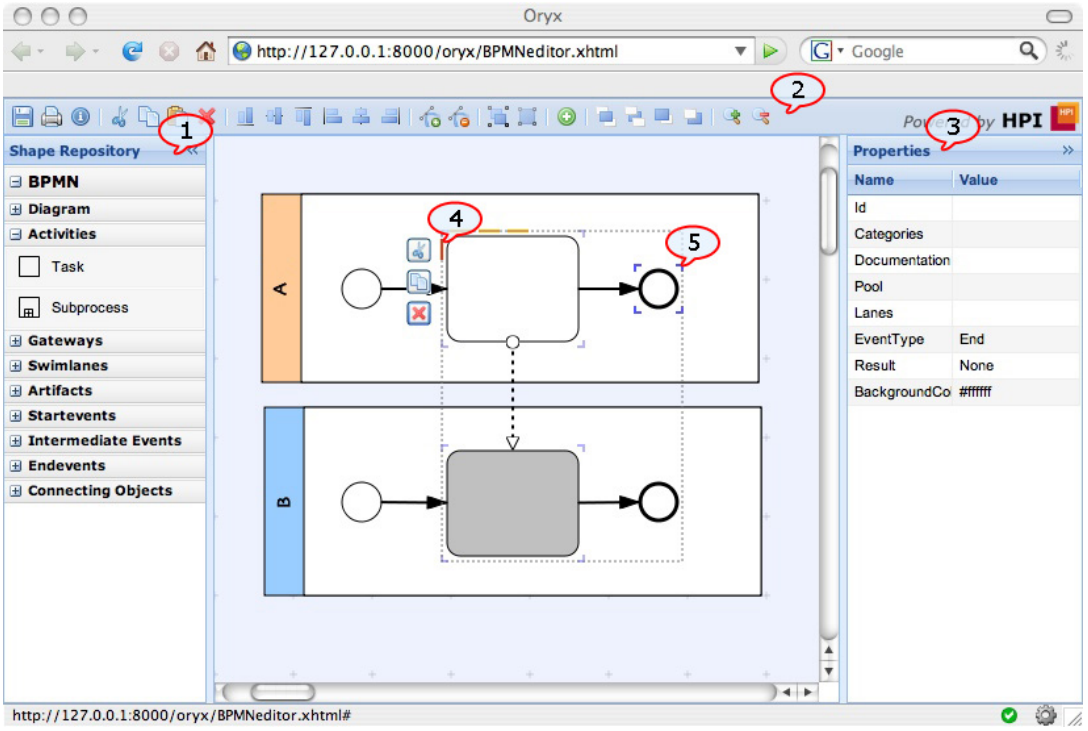

Abbildung 5.2: Screenshot - Editor

Edit

Dieses Plugin bietet die Möglichkeit zum Löschen, Ausschneiden, Kopieren und Einfügen von Objekten an. Beim Einfügen wird das betreffende Objekt neu erstellt, alle Attribute werden kopiert und die Bounds wird etwas nach unten versetzt.

File

Das File-Plugin bietet zusätzlich zur Möglichkeit des Speicherns ein Info-Fenster für den Editor an. Des Weiteren wird das Drucken des aktuellen Prozesses angeboten (Anforderung 2.1.18). Dabei wird aus der Seite der komplette SVG-DOM kopiert und in einem neuen Fenster eingefügt. Danach wird es auf eine DIN A4 Seite transformiert und der Druck-Dialog geöffnet.

View

Das View-Plugin bietet eine Zoom-Funktion an. Zum einem wird der Zoom-Faktor des SVG-Elements gesetzt und zum anderen wird die Größe der Canvas entsprechend angepasst.

Arrangement

Abbildung 5.3: Arrangement

Dieses Plugin bietet Funktionalität zum Ausrichten eines Objektes an. Entweder können Objekte an der Position anderer Objekte oder auf der Z-Achse ausgerichtet werden. Die Ausrichtung an der Position wird wie folgt angeboten: oben, unten, rechts, links und im Zentrum. Die Ausrichtung auf der Z-Achse sieht wie folgt aus: ganz nach vorne, eins nach vorne, eins nach hinten und ganz nach hinten.

Grouping

Dieses Plugin realisiert eine Gruppierungs-Funktion. Es reagiert auf Selektionsänderungen und fügt Elemente hinzu, welche sich in der gleichen Gruppe des selektierten Elements befinden.

AddDocker

Das Plugin AddDocker fügt einer Kante einen Docker an der Mouse-Down Position hinzu, nachdem diese Funktionalität aktiviert wurde. Zusätzlich können auch Docker wieder entfernt werden.

DragDocker

Dieses Plugin realisiert das Drag und Drop eines Dockers. Dabei wird überprüft, über welchem aktuellen Shape sich dieser Docker befindet, und es wird signalisiert, ob er sich an das Shape andocken kann. Die Information darüber entnimmt er dem dafür relevanten Stencil Set.

DragDropResize

Das DragDropResize-Plugin bietet die Möglichkeit, ein Shape entweder zu verschieben oder seine Größe zu ändern. Um diesen Vorgang besser zu signalisieren, wird ein Selektionsrahmen um alle selektierten Shapes gezogen. Beim Draggen werden zusätzlich die Shapes am Grid und an allen anderen Shapes ausgerichtet (solange die STRG-Taste nicht gedrückt wird).

Shape-Menu

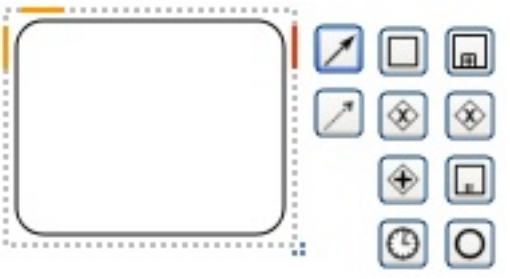

Abbildung 5.4: Shape-Menu

Das Shape-Menu bietet relevante Funktionalität am aktuellen Selektionsrahmen an (siehe Abbildung 5.2, Punkt 4). Sie realisiert somit die Anforderung 2.1.17. Die Information darüber, ob die Funktionalität relevant ist, wird aus den Meta-Daten der Plugins entnommen. Dort kann z.B. angegeben sein, ob eine Funktionalität verfügbar ist, wenn nur ein Shape selektiert ist oder wenn mehrere Shapes selektiert sind.

Des Weiteren bietet das Shape-Menu als Möglichkeit an, Folge-Shapes direkt am Shape zu erzeugen. Es werden ausschließlich gültige Verbindungen angezeigt, die im Stencil Set angegeben sind. So kann z.B. von einem Knoten direkt eine Kante neu erzeugt werden, wodurch beide mit einander Verbunden sind.

Toolbar

Die Toolbar bietet alle Funktionalitäten als Button im oberen Bereich des Editors an (siehe Abbildung 5.2, Punkt 2). Sie überprüft, ob die jeweilige Funktionalität verfügbar ist.

PropertyWindow

Das PropertyWindow zeigt alle Einstellungen und Eigenschaften des selektierten Shapes an (siehe Abbildung 5.2, Punkt 3). Durch Ändern der Werte im Property-Window werden auch die Eigenschaften des selektierten Shapes angepasst.

ShapeRepository

Das ShapeRepository zeigt alle Prozesselemente der geladenen Stencil Sets an (siehe Abbildung 5.2, Punkt 1), dadurch wird die Anforderung 2.1.10 umgesetzt. Dabei wird jedes Stencil Set in einer extra Gruppe angezeigt, die wiederum die Stencils gruppieren. Die Gruppenzugehörigkeit der Stencils können im Stencil Set angegeben werden.

Aus dem Shape Repository heraus können die Stencils mittels Drag und Drop auf der Zeichenfläche erzeugt werden. Dabei wird überprüft, ob das neue Shape auch in dem darunter liegenden Shape enthalten sein darf. Dies wird grafisch angezeigt.

SelectionFrame

Mittels Drag und Drop auf der Zeichenfläche kann ein Bereich aufgespannt werden, dem alle enthaltenen Elemente der Selektion hinzugefügt werden.

ShapeHighlighting

Dieses Plugin realisiert das Highlighting von Shapes. Durch ein Event kann dieses Plugin angesprochen werden. Dabei werden alle angegebenen Elemente mit der übergebenen Farbe markiert. Ein Beispiel einer Markierung eines Shapes wird in der Abbildung 5.2 im Punkt 5 gezeigt.

AddStencilSet

Das Laden eines neuen Stencil Sets wird durch diese Plugin angeboten (siehe Anforderung 2.1.12). Es muss ein relativer oder absoluter Pfad übergeben werden.

Loading

Um ein besseres Benutzerfeedback zu bekommen, realisiert dieses Plugin einen „Loading...“-Banner. Dies kann bei längeren Wartezeiten durch ein Event ein- und ausgeschaltet werden.

6. Ausblick

Abschließend wird noch beschrieben, welche Anforderungen der Editor nicht erfüllt hat und welche Erweiterungen sinnvoll wären.

Ein Problem des Editors war, dass es keine Anforderung für ein Backend gab, d.h. für einen Server, der die Datenhaltung übernimmt. Die Implementierung des Servers wurde vom Bachelorprojekt Thanis durchgeführt, die Integration beider Projekte wurde nicht ganz vollendet. Damit sind einige Anforderungen seitens des Editors nicht erfüllt, da die Grundlage zum Teil nicht geschaffen war. Nachstehend werden alle Anforderungen angegeben, die nicht oder nur zum Teil erfüllt wurden.

Vorhandene Prozesse (2.1.11) werden nicht im Shape Repository angezeigt. Der Grund dafür ist, dass es zur Zeit noch keine Möglichkeit gibt, dem Server eine Anfrage zu stellen, welche Prozesse bereits vorhanden sind.

Versionierung und Collaboration (2.1.13) sind nicht umgesetzt, da das Backend diese Funktionalität noch nicht zur Verfügung stellt.

UI Personalisierung (2.1.16) ist nur zum Teil implementiert. So ist es möglich, die Benutzerschnittstelle an den jeweiligen Benutzer anzupassen. Es ist aber nicht möglich, diese Einstellungen zu speichern. Dieses hat historische Gründe, da die Zeichenfläche nicht als Ressource auf dem Server gespeichert wird. Somit können die Einstellungen nicht persistent abgelegt werden.

Exportfunktion (2.1.19) ist nicht umgesetzt. Diese sollte nach jetzigem Standpunkt eher auf der Server-Seite realisiert werden. Da der Editor komplett ressourcenorientiert arbeitet, würde es eher Sinn machen, eine URL auf eine andere Repräsentation des Prozesses zeigen zu lassen, wodurch z.B. ein PNG, also eine Bitmap-Grafik, zurückgeliefert wird.

Folgend werden einige Erweiterungen aufgelistet, die sinnvoll für den Editor wären:

Verfügbarkeit: Der Editor sollte im nächsten Schritt auf alle Browser portiert werden, um eine höhere Akzeptanz zu schaffen.

Layouting: Der Editor sollte gewisse Layouting-Algorithmen implementieren. So sollte z.B. ein Layouting für Kanten realisiert werden, bei dem sich Kanten um Elemente herum platzieren.

Word-Wrap: Text in Prozesselementen könnten sich automatisch an der Größe der Elemente orientieren, um zu gewährleisten, dass sich Text nur innerhalb eines Objektes befindet.

A. Events

A.1 Auftretende Events im Editor

Alle Events, an denen man sich im Editor registrieren kann, werden in der Tabelle A.1 aufgelistet. Diese können auch mit gleichen Argumenten geworfen werden.

Event-Typ	Argumente	Beschreibung
'mousedown'	window.event, ORYX.Core.UIObj	Mouse-Down Event, welches von den Elementen im Editor weitergeleitet wurde. Es wird das Event-Objekt und das zugehörige UIObj übergeben.
'mousemove'	window.event, ORYX.Core.UIObj	Mouse-Move Event, welches von den Elementen im Editor weitergeleitet wurde. Es wird das Event-Objekt und das zugehörige UIObj übergeben.
'mouseup'	window.event, ORYX.Core.UIObj	Mouse-Up Event, welches von den Elementen im Editor weitergeleitet wurde. Es wird das Event-Objekt und das zugehörige UIObj übergeben.
'mouseover'	window.event, ORYX.Core.UIObj	Mouse-Over Event, welches von den Elementen im Editor weitergeleitet wurde. Es wird das Event-Objekt und das zugehörige UIObj übergeben.
'mouseout'	window.event, ORYX.Core.UIObj	Mouse-Out Event, welches von den Elementen im Editor weitergeleitet wurde. Es wird das Event-Objekt und das zugehörige UIObj übergeben.

'keydown'	window.event	Key-Down Event, welches im Editor geworfen wird.
'selectionchanged'	{type:'selectionchanged', elements:ORYX.Core.Shape[], subSelection:ORYX.Core.Shape}	Event für die Selektionsänderung im Editor. Übergeben wird ein Objekt mit den Elementen und einem möglichen Element für eine Subselektion. Zur Vereinfachung reicht es aus, wenn ein Plugin die Methode „onSelectionChanged“ implementiert. Diese wird bei Selektionsänderung aufgerufen.
'stencilSetLoaded'	{type:'stencilSetLoaded'}	Event, welches geworfen wird, wenn ein neues Stencil Set geladen wurde.
'loading.enable'	{type:'loading.enable'}	Event, das den Ladezustand aktiviert, der 'Loading...'-Banner wird sichtbar
'loading.disable'	{type:'loading.disable'}	Event, welches den Ladezustand deaktiviert.
'dragdrop.start'	{type:'dragdrop.start'}	Event, welches geworfen wird, wenn Drag und Drop gestartet wird.
'dragdrop.end'	{type:'dragdrop.end'}	Event, das bei Beendigung von Drag und Drop geworfen wird.
'highlight.showHighlight'	{type:'highlight.showHighlight', highlightId:string, elements:ORYX.Core.Shape[], color:#rrggbb, opacity:float}	Aktivierung des Highlighting von Elementen. Übergeben wird ein Objekt, welches die Id des Highlighting beinhaltet, die Elemente die hervorgehoben werden soll und die Farbe und den Transparenzanteil der für das Highlighting gesetzt werden soll.
'highlight.hideHighlight'	{type:'highlight.hideHighlight', highlightId:string}	Deaktivierung des Highlighting.

Abbildung A.1: Vorkommende Events im Editor

Literaturverzeichnis

[1] Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG), February 2006. http://www.bpmn.org/.

[2] Opera Software ASA. SVG in support in Opera 9. Available at: http://www.opera.com/docs/specs/opera9/svg/, 2007.

[3] IBM Corporation. Design basics. Available at: http://www-03.ibm.com/easy/page/6, 2005.

[4] Martin Czuchra. Oryx - Embedding Business Process Data into the Web. Hasso-Plattner Institute, Potsdam, Germany, 2007.

[5] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. Available at: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, 2000.

[6] Mozilla Foundation. SVG in Firefox. Available at: http://developer.mozilla.org/en/docs/SVG_in_Firefox, 2006.

[7] Joshua Gertzen. Object Oriented Super Class Method Calling with JavaScript. Available at: http://truecode.blogspot.com/2006/08/object-oriented-super-class-method.html, 2006.

[8] SVG Working Group. Scalable Vector Graphics (SVG) 1.1 Specification. Available at: http://www.w3.org/TR/2003/REC-SVG11-20030114/, 2003.

[9] Web Hypertext Application Technology Working Group Ian Hickson. HTML 5, Working Draft. Available at: http://www.whatwg.org/specs/web-apps/current-work/, 2007.

[10] Adobe Systems Incorporated. Getting Started with Flex 2. Available at: http://download.macromedia.com/pub/documentation/en/flex/2/flex2_gettingstarted.pdf, 2006.

[11] Nicolas Peters. Oryx - Stencil Set Specification. Hasso-Plattner Institute, Potsdam, Germany, 2007.

[12] Daniel Polak. Oryx - BPMN Stencil Set Implementation. Hasso-Plattner Institute, Potsdam, Germany, 2007.