Erstellung eines Prototypen zur nutzerfreundliehen Integration des Widget-Nachrichtenaustauschs in User-Interface-Mashups

Christian Fischer

Technische Universität Chemnitz

christian.fischer@cs.tu-chemnitz.de

Abstract: Das Konzept der Inter-Widget-Kommunikation ermöglicht einen Nachrichtentransfer zwischen unterschiedlichen Widgets und kann damit die Endbenutzer-Entwicklung signifikant vereinfachen. In diesem Paper wird ein Prototyp zur Realisierung des Widget-Nachrichtenaustauschs in Mashups auf der Basis von Apache RAVE, Apache Wookie und Apache Shindig beschrieben. Dieser ermöglicht es, dass Nachrichten zwischen Widgets mittels des OpenAjax Hub auf einfache Art und Weise ausgetauscht werden können - ohne die originalen Widgets umständlich verändern zu müssen. Dieses Dokument bezieht sich somit auf ein vorhergehendes Paper „Konzepte zur Integration des Widget-Nachrichtenaustauschs in User-Interface-Mashups“, wo verschiedene Konzepte zur Realisierung vorgestellt wurden.

Keywords: IWC, Widget, Gadget, Mashup, Rave, Wookie, Shindig

1 Einleitung

In sog. User-Interface-Mashups können die Nutzer ihre benötigten Informationen auf einfache Art und Weise durch die individuelle Kombination verschiedener Widgets aggregieren. Um in solchen Mashups die Produktivität zu steigern und das Nutzererleben zu optimieren, kann das Konzept der Inter-Widget-Kommunikation (IWC) angewandt werden. Dieses erlaubt einen Nachrichtentransfer zwischen verschiedenen Widgets und kann somit die Endbenutzer-Entwicklung signifikant vereinfachen.

Das Problem besteht bei der Implementierung der Inter-Widget-Kommunikation. Die Widget-Entwickler müssten ihre Widgets bereits so programmieren, dass die Nutzung der IWC-Technik möglich wird. Da es aber viele verschiedene Techniken gibt, ist nur eine spezialisierte Implementierung von Seiten des Entwicklers aus möglich. Daher wird in diesem Paper ein allgemein gültigeres Konzept genutzt, welches nicht die Anpassung der einzelnen Widgets fokussiert, sondern eine Implementierung seitens des Containers, der die verschiedenen Widgets bereitstellt. Die originalen Widgets sollen daher so wenig wie möglich angepasst werden. Falls es dennoch notwendig sein sollte, ein Widget

anzupassen, so muss dies möglichst automatisch geschehen, ohne dass der Nutzer im Quellcode umständliche Ergänzungen vornehmen muss. Es wird damit eine nutzerfreundliche Implementierung der IWC in User-Interface-Mashups angestrebt. Ziel ist es weiterhin, Benutzereingaben so zu verarbeiten, dass eine automatisierte Zustandssynchronisation anhand der individuellen Nutzereingaben stattfindet.

In Kapitel 2 werden zunächst die genutzten Anwendungen und Grundlagen vorgestellt. Die Erstellung des Prototyps wird in Kapitel 3 erläutert. Abschließend folgen eine Zusammenfassung und der Ausblick.

2 Apache Rave, -Wookie, -Shindig, Features und OpenAjax Hub

Um ein Mashup aus unterschiedlichen Widgets erstellen zu können, benötigt man eine Mashup-Plattform, die aus verschiedenen Containern Widgets arrangieren und anzeigen kann. Apache Rave[1] ist eine Mashup-Plattform, die es ermöglicht W3C-Widgets[2] mittels Apache Wookie[3] und OpenSocial-Widgets[4] durch Apache Shindig[5] für den Nutzer in Form eines Mashups bereitzustellen.

Apache Rave bietet ein Nutzermanagement, wodurch es möglich ist, für jeden Nutzer individuelle Mashups zu erstellen. Ein Widgetstore zum einfachen Hinzufügen beliebiger OpenSocial- oder W3C-Widgets aus Apache Wookie ist ebenfalls vorhanden. Für den Nutzer stellt Apache Rave die zentrale Interaktionsschnittstelle mit den individuellen Mashups des Nutzers dar. Im Wesentlichen kann der Nutzer Widgets hinzufügen, diese im Mashup positionieren und natürlich auch wieder entfernen. Zusätzlich ist es möglich, verschiedene Seiten zu definieren, wobei jede Seite wie ein eigenes Mashup betrachtet werden kann. Für verschiedene Situationen können somit entsprechend angepasste Mashups erstellt werden.

Widgets, welche durch Apache Wookie oder Apache Shindig bereitgestellt werden, können Gebrauch von sog. Features machen. Features sind Erweiterungen der Widget-API, wodurch ein erweiterter Funktionsumfang für die Widgets zur Verfügung steht. Ein wichtiges Feature ist der OpenAjax Hub[6]. Dieser ermöglicht u. a. den sicheren Nachrichtentransfer zwischen geschützten iframes (Cross Domain Policy). Da Widgets in Apache Rave innerhalb von iframes dargestellt werden um ein gewisses Maß an Sicherheit zu bieten, ist dieses Feature von besonderer Bedeutung für die Implementierung der IWC. Apache Rave unterstützt den OpenAjax Hub, indem spezielle OpenAjax iframes genutzt werden.

---

[1] siehe http://rave.apache.org/
[2] siehe http://www.w3.org/TR/widgets/
[3] siehe http://incubator.apache.org/wookie/
[4] siehe http://opensocial-resources.googlecode.com/svn/spec/2.5/OpenSocial-Specification.xml
[5] siehe http://shindig.apache.org/
[6] siehe http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification

3 Erstellung des Prototyps

3.1 Anforderungen

Es sollen auf einfache Art und Weise gewünschte Daten aus HTML Formularelementen (Input, Select, etc.) von einem Widget in die Formularelemente anderer Widgets transferiert werden. Dies soll für den Nutzer einfach zu definieren sein bzw. vollautomatisch während der Benutzung des Mashups geschehen. Vorhandene Widgets sollen nicht oder nur falls es unbedingt notwendig ist automatisch angepasst werden. Für jeden Nutzer des Mashups sind die individuellen Einstellungen zu speichern, sodass auch nach Schließen einer Session die Einstellungen wieder verfügbar sind. Die Implementierung soll durch eine Kombination aus Container- und Helperwidget-Konzept geschehen (siehe [1]).

3.2 Umsetzung

Die Kommunikation zwischen den Widgets soll mit Hilfe des OpenAjax Hubs geschehen. Apache Rave, Apache Shindig und Apache Wookie bieten die Unterstützung des OpenAjax Hubs, falls das Feature im Widget definiert wurde. Für die Umsetzung des Prototyps ist sicherzustellen, dass dieses Feature stets automatisch aktiviert wird. Die standardmäßige Aktivierung des OpenAjax Hubs für alle Widgets ist jedoch von Apache Rave, Wookie und Shindig nicht vorgesehen. Es ist daher eine Anpassung dieser Anwendungen dahingehend notwendig.

Die Erkennung von Nutzereingaben kann am elegantesten über das Hinzufügen eines Features zu den Widget-Containern realisiert werden – es wird daher ein neues Feature für Apache Wookie und Apache Shindig erstellt, welches die Nutzereingaben erkennt und entsprechend über den OpenAjax Hub weiterleitet. Es wird außerdem ein neues W3C-Widget (Helper-Widget) erstellt, welches die zentrale Logik und dem Nutzer weitere Informationen bereitstellt. Durch die Integration der zentralen Logik mit Hilfe eines Widgets soll sichergestellt werden, dass die Portierung auf andere Mashup-Plattformen vereinfacht wird. Der Nutzer hat zudem jederzeit die Möglichkeit, die erweiterte Funktionalität durch das Entfernen des Helper-Widgets zu deaktivieren.

3.2.1 Anpassung von Apache Wookie

Nachfolgend werden die Änderungen in Apache Wookie beschrieben. In der Datei `StartPageProcessor.java` (siehe Abbildung 1) werden u. a. die Features geprüft und die dazugehörigen Dateien geladen (JavaScript, CSS, etc.). Damit in jedem Widget außer dem Master-Widget (Helper-Widget) die erweiterte Funktionalität (OpenAjax Hub, IWC-Erweiterung) zur Verfügung steht, muss die Funktion `addFeatures` angepasst werden (siehe Abbildung 1).

```
\wookie-snapshot\src\org\apache\wookie\util\html\StartPageProcessor.java

private void addFeatures(IHtmlProcessor engine,W3CWidget model) throws Exception{
        // IWC-Prototype Start
        boolean isopenajax = false, ismaster = false;
        // IWC-Prototype End

        for (org.apache.wookie.w3c.IFeature feature: model.getFeatures()){
            // IWC-Prototype Start
            if(feature.getName().equals("http://www.iwc.org/master"))ismaster=true;
            if(feature.getName().equals("http://www.openajax.org/hub"))isopenajax=true;
            // IWC-Prototype End
            for (IFeature theFeature: Features.getFeatures()){
              if (theFeature.getName().equals(feature.getName())){
                  addScripts(engine, theFeature);
                  addStylesheets(engine, theFeature);
              }
            }
        }

        // IWC-Prototype Start
        // IF NOT OpenAjax Hub is requested - Insert OpenAjax Hub Support Code
        if(!isopenajax){
            for (IFeature theFeature: Features.getFeatures()){
              if (theFeature.getName().equals("http://www.openajax.org/hub")){
                  addScripts(engine, theFeature);
                  addStylesheets(engine, theFeature);
              }
            }
        }
        // IF NOT Master Widget - Insert Client Feature
        if(!ismaster){
            for (IFeature theFeature: Features.getFeatures()){
              if (theFeature.getName().equals("http://www.iwc.org/client")){
                  addScripts(engine, theFeature);
                  addStylesheets(engine, theFeature);
              }
            }
        }
        // IWC-Prototype End
    }
```

Abbildung 1: StartPageProcessor.java

Die derzeitige Implementierung des OpenAjax Hubs in Apache Wookie hat den Nachteil, dass der Hub nicht sofort nach Erstellung eines Widgets bereit ist, um Daten entgegen zu nehmen bzw. Daten zu senden. Das Originalfeature wurde entsprechend angepasst, damit die Verbindung zum OpenAjax Hub manuell mit Hilfe einer Callback-Funktion hergestellt wird. Sobald der Hub bereit ist, wird die entsprechende Callback-Funktion aufgerufen. Standardmäßig wird automatisch eine Verbindung zum Hub hergestellt – jedoch ohne Callback-Möglichkeit.

```
<feature>
    <name>http://www.openajax.org/hub</name>
    <script src="OpenAjaxManagedHub-all.js"/>
    <!--
    Manually connect to the Hub with Callback to ensure that the Hub is
ready!
    <script src="OpenAjaxClient.js"/> -->
</feature>
```

Abbildung 2: feature.xml des OpenAjax Hubs

Weiterhin wurden die zwei Features *iwc-client* und *iwc-master* in Apache Wookie durch das Anlegen der Ordner inkl. der nötigen Definitionsdatei `feature.xml` und den JavaScript-Dateien im Verzeichnis `wookie-snapshot\features` hinzugefügt.

3.2.2 Anpassung von Apache Shindig

Bei Apache Shindig ist das Vorgehen ähnlich wie bei Apache Wookie. Es wird ein neues Feature `iwc-client` erstellt und durch den Eintrag

```
shindig.gadget-rewrite.default-forced-libs=core:rpc:iwc-client
```

in der Datei `\java\common\conf\shindig.properties` standardmäßig geladen. Es ist außerdem notwendig, in der Datei `gadget_holder.js` immer den OpenAjax-iframe zu laden, da andernfalls keine Kommunikation mit dem Hub möglich ist (siehe Abbildung 3).

```
\features\src\main\javascript\features\container.site.gadget\gadget_holder.js

osapi.container.GadgetHolder.prototype.render = function(gadgetInfo,
viewParams, renderParams) {
  this.iframeId_ = osapi.container.GadgetHolder.IFRAME_ID_PREFIX_ +
      this.site_.getId();
  this.gadgetInfo_ = gadgetInfo;
  this.viewParams_ = viewParams;
  this.renderParams_ = renderParams;

// IWC-Prototype Start

//  if (this.hasFeature_(gadgetInfo, 'pubsub-2')) {
//    this.doOaaIframeHtml_();
//  } else {
//    this.doNormalIframeHtml_();
//  }

  this.doOaaIframeHtml_();

// IWC-Prototype End

};
```

Abbildung 3: gadget_holder.js; standardmäßiges Laden des OpenAjax-iframes

Das neue `iwc-client` Feature wird durch das Hinzufügen des Ordners `iwc-client` mit der entsprechenden Definitionsdatei `feature.xml` im Ordner

`\extras\src\main\javascript\features-extras`

und dem Hinzufügen einer Zeile mit dem Inhalt

`features-extras/iwc-client/feature.xml`

in der Datei `features.txt` im gleichen Ordner realisiert. Der Inhalt der `feature.xml`-Datei ist in Abbildung 4 zu sehen. Als Abhängigkeit wird das pubsub-2 Feature, welches bei Apache Shindig dem OpenAjax Hub entspricht, angegeben. Damit wird sichergestellt, dass der Hub genutzt werden kann. Zudem wird jQuery und die entsprechende IWC-Erweiterung geladen.

```
<?xml version="1.0"?>
<feature>
  <name>iwc-client</name>
  <dependency>globals</dependency>
  <dependency>pubsub-2</dependency>
  <gadget>
    <script src="jquery.js" />
    <script src="iwc-client.js" />
  </gadget>
</feature>
```

Abbildung 4: feature.xml; Shindig iwc-client Feature

3.2.3 Anpassung von Apache Rave

In Apache Rave selbst muss nicht viel verändert werden. Es müssen lediglich die CSS- und JavaScript-Dateien verlinkt werden, die auf der Portalseite dem Anzeigen der Informationen dienen. Für CSS-Dateien geschieht dies in der Datei `custom_css.tag` (siehe Abbildung 5). Es muss natürlich beim Erstellen und Konfigurieren von Apache Rave darauf geachtet werden, dass die angepassten Apache Wookie- und Apache Shindig-Container anstatt der automatisch genutzten Original-Container genutzt werden.

```
\rave-project\rave-portal-resources\src\main\webapp\WEB-INF\tags\custom_css.tag

<!– IWC-Prototype Start -->

<link rel="stylesheet" href="/portal/static/css/jquery-ui.css"
      type="text/css" media="all" />
<link rel="stylesheet" href="/portal/static/css/jquery.gritter.css"
      type="text/css" media="all" />

<!-- IWC-Prototype End -->
```

Abbildung 5: Apache Rave custom_css.tag

Die JavaScript-Dateien werden durch die Datei `custom_js.tag` hinzugefügt (siehe Abbildung 6). Hier wird nur ein jQuery-Plugin geladen. Die restliche Logik befindet sich komplett in den beiden Features *iwc-client* und *iwc-master* und sind somit losgelöst von der verwendeten Portalseite, welche das Mashup bereitstellt (hier Apache Rave).

```
\rave-project\rave-portal-resources\src\main\webapp\WEB-INF\tags\custom_js.tag

<!– IWC-Prototype Start -->

<script type="text/javascript"
src="/portal/static/script/iwc/jquery.gritter.min.js"></script>

<!-- IWC-Prototype End -->
```

Abbildung 6: Apache Rave custom_js.tag

3.2.4 Implementierung der automatisierten Inter-Widget-Kommunikation

Die beiden genannten Features *iwc-client* und *iwc-master* stellen die Hauptfunktionalität der automatisierten IWC dar. Das Feature *iwc-client* stellt sicher, dass Nutzereingaben erkannt werden und leitet die eingegebenen Daten und Aktionen (Button Click, etc.) mittels des Publish-Subscribe-Verfahrens des OpenAjax Hubs zum Helper-Widget weiter. Jedes Widget innerhalb des Mashups nutzt ein eigenes Thema (Topic) zum Senden und Empfangen von Daten. Damit sollen ggf. auftretende Seiteneffekte vermieden werden. Das Nachrichtenthema wird anhand der eindeutigen WidgetID von Apache Rave und dem HTML-Titel-Tag des Widgets generiert – dadurch soll zum einen eine eindeutige Zuordnung möglich sein und zum anderen sollen für den Nutzer lesbare Informationen generiert werden.

Das Helper-Widget lauscht hingegen auf allen Themen des Hubs (Thema: „ ** “). Es empfängt somit alle eingegebenen Daten und Aktionen des Benutzers. Diese werden daraufhin lokal gespeichert und verarbeitet. Technisch wird dies durch das Feature *iwc-master* realisiert, welches nur vom Helper-Widget genutzt wird. Die gesendeten Daten werden JSON-kodiert über den Hub gesendet.

Im Feature *iwc-master* erfolgt die Erkennung von Nutzereingaben anhand der letzten 10 Aktionen des Nutzers. Vorhergehende Aktionen werden der Übersicht halber und zur Vermeidung von Seiteneffekten nicht ausgewertet und aus Performancegründen auch verworfen. Es werden primär die letzten Änderungen in Eingabefeldern nach doppelt eingegebenen oder ausgewählten Daten durchsucht. Damit dies verständlich ist, hier ein kleines Beispiel: Ein Nutzer sucht in einem Map-Widget nach der Stadt „London“. Innerhalb der nächsten 10 Aktionen wählt er in einem Wetter-Widget auch die Stadt „London“ aus. Das System erkennt die doppelten Eingaben aus verschiedenen Widgets und speichert diese in einer Liste ab. Sucht der Nutzer nun erneut nach einer Stadt im Map-Widget, so wird dies erkannt und die soeben getätigte Eingabe wird direkt zum Wetter-Widget weitergeleitet. Eventuell nachfolgende Aktionen wie bspw. ein Button-Click kann ebenfalls automatisiert getätigt werden. Es ist dadurch möglich, unabhängig vom Nutzer diese Funktion zu implementieren und im Hintergrund die Erkennung laufen

zu lassen. Es ist aber ebenfalls möglich, das System vom Nutzer selbst für den jeweiligen Anwendungsfall erst einzulernen bevor es genutzt werden soll. Er/Sie muss dazu im Trainingsmodus, der standardmäßig aktiviert ist, beliebige doppelte Eingaben in den Formularfeldern tätigen, bis das System alle nötigen Quellen und Ziele gespeichert hat. Danach kann der Trainingsmodus deaktiviert und das Mashup damit genutzt werden. Neue doppelte Eingaben führen dann nicht mehr zum Weiterlernen des Systems. Das *iwc-master*-Feature sendet die nötigen Informationen über den OpenAjax Hub mittels der Widget-spezifischen Topics zu den richtigen Ziel-Widgets. Als Identifikator der HTML-Elemente innerhalb eines Widgets wird ein jQuery-Pfad zum Element genutzt, da dieser einfach und verständlich interpretiert werden kann. Dieser jQuery-Pfad wird im Feature *iwc-client* generiert und bei jeder Aktion mit übertragen. Es ist damit möglich, gezielt die entsprechenden Elemente zu adressieren. Damit können die gewünschten Daten und Aktionen eingefügt bzw. ausgeführt werden. Eingaben, die per Drag & Drop von Input-Elementen zu anderen Widgets ebenfalls in Input-Elementen geschehen, werden ebenso wie doppelte Eingaben betrachtet und verarbeitet.

3.2.5 Aufbau des Helper-Widgets

Das Helper-Widget besitzt mehrere Reiter, in denen der Nutzer Informationen beziehen und Einstellungen tätigen kann. Der Reiter „Log“ gibt Aufschluss über die aktuell empfangenen Daten des Helper-Widgets.

Der Reiter „Queues“ zeigt die aktuell gespeicherten Listen an. Unter anderem die letzten 10 Aktionen des Nutzers im Bereich „Current Events“. Wenn der Nutzer mit der Maus über die Listeneinträge fährt, werden die entsprechenden Elemente in den jeweiligen Widgets hervorgehoben um eine bessere Übersicht zu bekommen.

Im Bereich „Value + Click“ werden Aktionen gezeigt, welche nach der Dateneingabe bzw. der Änderung eines Eingabefeldes („change“) eine Aktion wie bspw. ein Klick auf einen Button beinhalten. Es ist zudem sofort erkennbar von welchem Widget zu welchem Widget die Daten gesendet werden (siehe Abbildung 7). Der Bereich „Double Values“ enthält nur Aktionen, die keine weitere Aktion nach dem Ändern eines Eingabefeldes nach sich führt – wo also die Daten zwar in den Zielfeldern eingetragen werden sollen, jedoch kein nachfolgender Klick auf einen Button o. Ä. erfolgen soll.

Der Bereich Settings enthält die Einstellungen des Widgets. Der Reiter Tests ist für Demonstrationszwecke vorgesehen. Hier kann bspw. die Funktionalität des OpenAjax Hubs mittels der Standard Apache Shindig OpenAjax Testwidgets überprüft werden.

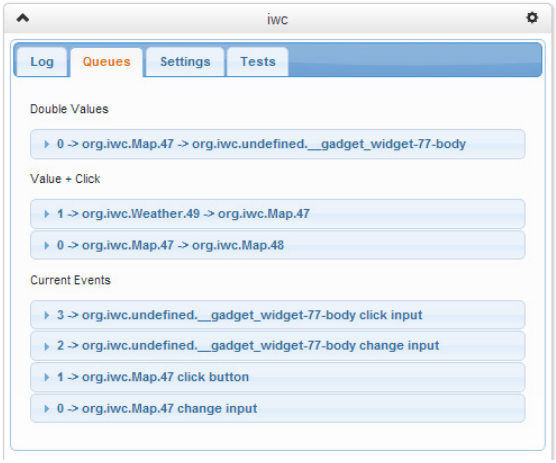

Abbildung 7: Ansicht des Helper Widgets, Queues

3.2.6 Einstellungen des Helper-Widgets

Das Helper-Widget bietet diverse Einstellungen, die es dem Nutzer ermöglichen,

- die gesamte IWC-Funktionalität ein-/auszuschalten (**IWC (Global)**),
- den Trainingsmodus ein-/auszuschalten (**Training-Mode**),
- zwischen den Modi „nur doppelte Eingaben“ (**Double-Value detection (insert only)**) und „doppelte Eingaben mit darauffolgender Aktion“ (**Double-Value detection (with click)**) umzuschalten und
- die Hervorhebung der Elemente im Reiter Queues ein-/auszuschalten (**Highlight Elements**, siehe Abbildung 8).

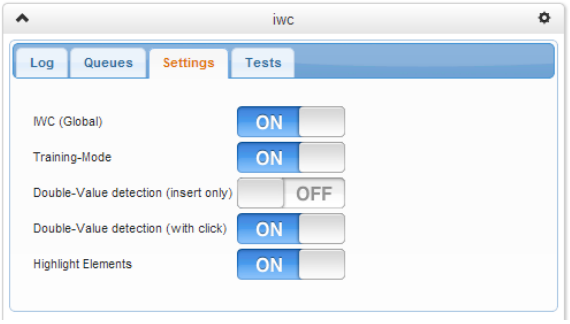

Abbildung 8: Ansicht des Helper Widgets, Settings

4 Fazit und Ausblick

Es wurde ein Prototyp erstellt, der es ermöglicht auf einfache Art und Weise für den Nutzer ein individuelles Setup des Widget-Nachrichtenaustauschs für ein Mashup in Apache Rave zu erstellen. Dies kann zum einen vollautomatisch während der Benutzung des Mashups oder individuell anhand des Einlernens durch den Nutzer geschehen. Im Wesentlichen werden doppelte Eingaben des Nutzers erkannt und verarbeitet – dadurch wird gewährleistet, dass ein Nutzer nicht umständlich das System trainieren muss, sondern auf einfache und intuitive Weise an die Benutzung des Features „Inter-Widget-Kommunikation“ herangeführt wird. Falls doppelte Eingaben erkannt werden, so wird dies dem Nutzer mittels eines unaufdringlichen Informations-Popups in der rechten oberen Ecke mitgeteilt.

Durch die Nutzung des OpenAjax Hubs wird ein sicherer Nachrichtentransfer zwischen den Widgets gewährleistet. Auch die Portierung auf ein anderes Mashup soll damit erleichtert werden. Die gezeigte Lösung der Implementierung mit Hilfe eines separaten Widgets bietet zudem die Möglichkeit die Funktionserweiterung auf einfache Art und Weise in ein Mashup zu integrieren oder zu entfernen.

Dieser Prototyp zeigt nutzerfreundlich, welche Möglichkeiten die IWC in Mashups bieten kann. Es werden derzeit INPUT, SELECT, TEXTAREA, BUTTON-Klicks und Drag&Drop Events verarbeitet. Um noch mehr Funktionalität bieten zu können, ist es durchaus denkbar diese Liste zu erweitern. Es könnte bspw. sinnvoll sein Hyperlink-Klicks oder Klicks innerhalb von Tabellen zu betrachten.

Durch die verschiedenen Möglichkeiten, Daten aus einem Formularfeld abzusenden, ist es nicht ohne Weiteres möglich, die nachfolgende Aktion nach einer Nutzereingabe zu erfassen und entsprechend wiederzugeben. Es werden bei diesem Prototyp Button-Klicks unterstützt, welche der normale Weg sein sollten, um Daten aus einem Formular zu senden. Weiterführend sollten daher mehrere mögliche Verfahren diesbezüglich implementiert werden. Mit diesem Prototyp wurden ausgewählte Konzepte aus [1] beispielhaft umgesetzt.

5 Literaturverzeichnis

[1] Christian Fischer, Olexiy Chudnovskyy, Martin Gaedke. "Unterstützung der Endbenutzer-Entwicklung mittels Inter-Widget-Kommunikation und User Interface Mashups". Chemnitz, Germany. *Studentensymposium Informatik Chemnitz.* [Online]. Verfügbar: http://vsr.informatik.tu-chemnitz.de/publications/2012/09. [Letzter Zugriff: 05.11.2012]