Konzepte zur Integration des Widget-Nachrichtenaustauschs in User-Interface-Mashups

Christian Fischer, Olexiy Chudnovskyy[1], Martin Gaedke[2]

Technische Universität Chemnitz
christian.fischer@cs.tu-chemnitz.de
olexiy.chudnovskyy@cs.tu-chemnitz.de
martin.gaedke@cs.tu-chemnitz.de

Abstract: Widgets findet man heutzutage auf immer mehr Webseiten oder Webportalen. Durch die individuelle Kombination verschiedener Widgets in Form eines Mashups ist es für einen Nutzer auf einfache Art und Weise möglich, die gewünschten Informationen zu aggregieren. Das Konzept der Inter-Widget Kommunikation ermöglicht einen Nachrichtentransfer zwischen verschiedenen Widgets und kann damit die Endbenutzer-Entwicklung signifikant vereinfachen. In diesem Paper wird ein Überblick über die Thematik gegeben und es werden mehrere Konzepte erläutert wie eine benutzerfreundliche Integration der Inter-Widget Kommunikation mit bereits vorhandenen Widgets möglich ist. Zudem werden auch andere Kommunikations-formen über die Inter-Widget Kommunikation hinaus angeschnitten.

Keywords: IWC, Widget, Gadget, Mashup, Web 2.0

1 Einleitung

Wenn man die heutige Nutzung des Internets mit der von vor einigen Jahren vergleicht, bemerkt man besonders in Zeiten des Web 2.0 ständige Veränderungen der Nutzungsformen. Die Benutzer aggregieren ihre benötigten Informationen nicht mehr ausschließlich durch das Besuchen einzelner Webseiten, sondern mehr und mehr durch Mashups. Die Anzahl der Mashups auf programmableweb.com belegt dies eindrucksvoll. Zum Zeitpunkt der Erstellung des Papers existieren auf diesem Portal 6655 Mashups mit einer Wachstumsrate von durchschnittlich 2,7 Mashups pro Tag[3].

Ein bekanntes Beispiel für ein Mashup ist das im Jahre 2007 entstandene iGoogle, ein freies Portal der Firma Google. Indem der Nutzer die sog. Google-Gadgets individuell arrangiert, wird für den jeweiligen Nutzer und Anwendungsfall die ideale Umgebung erstellt. Die Bedienung ist zudem intuitiv und einfach gestaltet.

[1] Wissenschaftlicher Betreuer
[2] Wissenschaftlicher Betreuer
[3] www.programmableweb.com/mashups , ausgehend vom 1.6.2012 über die letzten 6 Monate betrachtet

Die Kombination der Informationen für ein Mashup kann auf unterschiedlichen Wege realisiert werden. Es wird dabei zwischen Daten-Mashups und Benutzerschnittstellen-Mashups (UI-Mashups) unterschieden. [1] Bei den Daten-Mashups liegt der Fokus auf der Integration und Verarbeitung von Daten verschiedenster Quellen. RSS Feeds, XML und CSV Dateien sind nur ein paar Beispiele dafür. Yahoo!Pipes[4], JackBe Presto[5] sind beispielhafte Vertreter der Entwicklungsumgebungen dieser Art von Mashups. UI-Mashups legen dagegen den Fokus auf die Integration bereits vorhandener Benutzerschnittstellen aus dem Internet. Dies können HTML-Inhalte anderer Webseiten ebenso wie bereits existierende Widgets sein. [1] UI-Mashups verbergen die Komplexität der Technologie hinter den Widgets. Sie sind daher für die Endbenutzer-Entwicklung von besonderen Interesse und Grundlage für diese Arbeit.

Momentan erfordern die UI-Mashups eine manuelle Zustands-Synchronisation zwischen den Widgets. Dies ist jedoch ein entscheidender Nachteil für die Unterstützung der Endbenutzer-Entwicklung. Um diese bestmöglich zu unterstützen ist es notwendig, dass die Widgets Ihren Zustand selbstständig synchronisieren können. An dieser Stelle setzt die Inter-Widget-Kommunikation (IWC) an.

Mit Hilfe der IWC ist es möglich, einen Nachrichtenaustausch zwischen verschiedenen Widgets zu realisieren, sodass komplexere Szenarien implementiert werden können. Bspw. lässt sich der Zustand zwischen einem Wetter-Widget und einem Karten-Widget synchronisieren, wenn der Nutzer den Positionsmarker auf der Karte ändert. Je nach Wetterlage könnte bei diesem Beispiel durch die Rückmeldung des Wetter-Widgets auch ein Regenradar-Overlay auf der Karte angezeigt werden.

In diesem Paper werden Konzepte erarbeitet, die es ermöglichen Widgets eines Mashups mit der IWC-Funktionalität zu erweitern. Des Weiteren wird aus Sicht des Endbenutzers auf die Benutzerfreundlichkeit dieser Konzepte eingegangen. Das Ziel ist es ferner, die Endbenutzer-Entwicklung mittels UI-Mashups und Inter-Widget-Kommunikation zu unterstützen.

In Kapitel 2 werden zunächst die vorhandenen Konzepte und Grundlagen analysiert. Eigene Konzepte für eine Plattform zur Unterstützung der Endbenutzer-Entwicklung mittels UI-Mashups und IWC werden in Kapitel 3 erläutert, wobei jeweils auch auf die Benutzerfreundlichkeit der genannten Konzepte eingegangen wird. Abschließend folgt eine Zusammenfassung und der Ausblick.

2 Endbenutzer-Entwicklung mittels UI-Mashups und IWC

Im Folgenden werden die Grundlagen der Endbenutzer-Entwicklung mittels UI-Mashups vorgestellt. Zusätzlich wird ein kurzer Überblick bezüglich bereits vorhandener Ansätze gegeben.

[4] http://pipes.yahoo.com/pipes/
[5] http://www.jackbe.com

Der Begriff Mashup kommt ursprünglich aus dem musikalischen Bereich und beschreibt dort eine Zusammenführung mehrerer Lieder unterschiedlichen Stils zu einem neuen Werk. [2] Im Zusammenhang mit Widgets und dem World Wide Web beschreibt der Begriff die Aggregation und Aufbereitung verschiedenster Daten und Quellen zu einer zentralen Instanz, um damit einen Mehrwert zu bieten. [2] In diesem Kontext ist es demnach eine Webseite, die Informationen unterschiedlichen Ursprungs vereint und dem Nutzer bereitstellt.

Da in diesem Kontext Widgets eine entscheidende Rolle spielen, muss zunächst definiert werden, was genau mit dem Begriff Widget gemeint ist: Gemäß dem World Wide Web Consortium (W3C) ist ein Widget eine für Endbenutzer konzeptualisierte interaktive Minianwendung bezüglich eines speziellen Anwendungsfalls, um Daten aus dem Internet oder lokale Daten anzuzeigen bzw. zu aktualisieren. [3] Dabei muss die Minianwendung so gepackt sein, dass diese problemlos heruntergeladen und auf einem Computer, einem Mobiltelefon oder einem anderen internetfähigen Gerät installiert werden kann. [3]

Widgets werden innerhalb bestimmter Container, bspw. Apache Wookie[6], ausgeführt und können via Schnittstellen auf die Funktionalitäten dieser zugreifen. Diese Container können wiederum von Mashup-Plattformen wie Apache Rave[7] genutzt werden. Innerhalb der W3C Spezifikation gibt es keine Beschreibung dazu, wie Widgets miteinander kommunizieren können. Es ist daher notwendig, entweder andere Widget-Formate, welche IWC bereits beinhalten, als Grundlage für die weitere Arbeit zu nutzen oder die W3C Widgets entsprechend zu erweitern. Widgets werden in anderen Domänen auch als Gadgets bezeichnet.

Wenn mehrere Widgets miteinander kommunizieren sollen, so gibt es dafür mehrere Ansätze. Zum einen ist es relevant, welche Techniken für die Nachrichtenübertragung in Frage kommen, zum anderen ist es wichtig wie die Nachrichtenverteilung geschehen soll. Bezüglich der Kommunikation zwischen den Widgets können die Bereiche orchestrierte-, choreografierte- und hybride UI-Mashups klassifiziert werden. [1] Bei der orchestrierten Variante wird die gesamte Kommunikation über ein zentrales Element gesteuert. Dies ist vergleichbar mit einem Dirigenten eines Orchesters, daher auch der Name. In der choreografierten Variante kommunizieren hingegen die Widgets miteinander und tauschen die Informationen selbst aus. Vergleichbar mit einem Chor, der nur durch Zusammenarbeit ein gutes Ergebnis erhält. Die hybride Variante vereint die ersten beiden. Die Widgets dürfen demnach miteinander kommunizieren, es kann aber auch der Nachrichtenfluss zentral gesteuert werden.

Technisch gesehen können die auszutauschenden Nachrichten über mehrere Methoden transferiert werden. Eine Liste von Ivan Zuzak [4] gibt Aufschluss über die derzeit vorhandenen Projekte und Methoden bezüglich der Kommunikationsmöglichkeiten. Eine Klassifikation über die verschiedenen Möglichkeiten kann durch [4] wie folgt festgelegt werden:

[6] http://incubator.apache.org/wookie/
[7] http://rave.apache.org/

	Inter-Widget	Intra-Widget
Same-Browser	1	3
Cross-Browser	2	4

Tabelle 1: Matrix zur Klassifizierung der Widget-zu-Widget Kommunikation [5]

Mit Hilfe dieser Klassifikation und der genannten Liste lässt sich eindeutig eine Tendenz zu Kategorie 1 feststellen. Dies ist auch nicht verwunderlich, da dies einem üblichen Mashup entspricht. Eine Webseite beinhaltet viele Widgets und diese sollen ihre Nachrichten und Daten untereinander austauschen. In dieser Arbeit wird diese Kategorie daher näher betrachtet. Der Nachrichtenaustausch geschieht auf unterster Ebene meist mit dem HTML Standard Postmessage, welcher bspw. einen Nachrichtenaustausch innerhalb des Browserfensters zwischen mehreren iframes zulässt. [4] Zusätzlich sind sog. Publish-Subscribe (pubsub) Techniken etabliert. [4] In vielen Projekten von der Liste wird bspw. der OpenAjax Hub [5] genutzt. Bei den OpenSocial Gadgets und bei Apache Rave kann damit problemlos das pubsub-Konzept angewendet werden. [6] Wegen Google+ wurden diese Funktionen bei den iGoogle-Gadgets wieder entfernt. [7] Bei den W3C Widgets ist die Funktionalität der IWC derzeit nicht vorgesehen. [8] Das pubsub-Verfahren basiert auf Nachrichtenthemen (Topics) bzw. Nachrichtenkanäle (Channels). Eine Nachricht, die über einen Kanal veröffentlicht (published) wird, empfangen alle Abonnenten (subscriber). Auf diese Art ist eine lose Kopplung der beteiligten Komponenten möglich. Es kümmert also einen publisher nicht, ob sich jemand für die zu sendende Nachricht interessiert. Das ist im Kontext von Mashups und Widgets insoweit von Vorteil, als dass dadurch problemlos Widgets ausgetauscht werden können.

Plattformen wie bspw. scrapplet.com oder das Apache Rave Projekt besitzen eine Implementierung des pubsub-Verfahrens durch den OpenAjax Hub. Es muss jedoch manuell JavaScript Code innerhalb der Widgets ergänzt werden, damit diese erweiterte Funktionalität auch genutzt werden kann. Das ist ein klarer Nachteil dieser Plattformen. Die Nutzbarkeit für den Endbenutzer hinsichtlich der IWC ist damit eingeschränkt. Derzeit ist es für einen Endbenutzer nicht ohne weiteres möglich bereits existierende Widgets, die keine IWC Funktionalität aufweisen, IWC-fähig zu machen. Anders als für die Endbenutzer-Entwicklung, stellt es für einen Entwickler kein Problem dar, da dieser seine eigenen Widgets mit dem erweiterten Funktionsumfang neu erstellen kann. Dabei gibt es bereits Ansätze um die Integration der IWC in einem Mashup für den Anwender benutzerfreundlicher zu gestalten: Bspw. gibt es Bemühungen und ein erstes Framework um Drag & Drop Funktionalitäten in Mashups zu integrieren. [9] Generell sind aber die vorhanden Lösungen sehr domänenspezifisch und daher für die allgemeine Endbenutzer-Entwicklung kaum zu gebrauchen. Bei iGoogle oder den Windows-Minianwendungen sucht man IWC derzeit vergebens.

Ein anderer Ansatz [10] bestrebt vorhandene Webstandards-basierte Widgets in Richtung IWC zu erweitern. Dies funktioniert sehr gut, jedoch erfordert diese Methode Hintergrundwissen von IWC-Konzepten und Semantic-Web Technologien. Ein

Systemverwalter kann damit, ohne selbst den Quellcode anzupassen, IWC-fähige Widgets für die Nutzer bereitstellen. Für die Endbenutzer-Entwicklung können diese Widgets entsprechend unkompliziert genutzt werden. Ohne das entsprechende Hintergrundwissen kann der Nutzer jedoch selbst keine Widgets bereitstellen. Da bei dieser Methode der Container angepasst wurde, ist eine Änderung des Quelltextes nicht notwendig. Das ist ein wichtiger Schritt in die richtige Richtung.

3 Erweiterung der UI-Mashups zur Unterstützung von IWC

Im Folgenden werden verschiedene Konzepte für die Unterstützung der Endbenutzer-Entwicklung mittels IWC genannt. Es werden jeweils die nötigen Voraussetzungen und die beabsichtigte Wirkung des Konzepts erläutert.

3.1 Containerkonzept

Besteht die Möglichkeit den Widget-Container zu beeinflussen oder durch Plug-Ins zu erweitern, steht diese Herangehensweise zur Verfügung. Der Container, in dem die Widgets gerendert und ausgeführt werden, stellt bei diesem Konzept eine zentrale JavaScript Instanz (im folgenden JS-Helper genannt) zur Verfügung. Sofern es durch den Container technisch möglich ist, durchsucht der JS-Helper alle Widgets innerhalb des Containers nach Formularelementen. Um auf bestimmte Ereignisse und Aktionen innerhalb der Widgets reagieren zu können, werden jeweils Event-Listener hinzugefügt. Die ausgeführten Aktionen werden vom JS-Helper gespeichert. Er lernt von den gewählten Aktionen des Nutzers, bei denen jeweils auch eine Kategorie durch ein Dialogsystem abgefragt wird. Die Kategorien wie bspw. Websuche, Wetter, Standort, usw. sollten thematisch abgegrenzt sein. Zusammen mit einem Wörterbuch ist es bei einer weiteren Abfrage durch die Kategorisierung möglich, mit gleichen oder ähnlichen Eingaben die hinterlegten Aktionen durchzuführen. Das Wörterbuch sollte dabei bereits einen gewissen Wortstamm mit zugehöriger Kategorisierung aufweisen. Dies kann auch durch einen zentralen Webdienst realisiert werden.

Die gespeicherten Aktionen sollen auch beim nächsten Besuch des Mashups wieder verfügbar sein. Cookies oder ein Loginsystem des Containers sind denkbare Speichermöglichkeiten. Der Nutzer muss zudem jederzeit die Möglichkeit haben, das Unterstützungssystem zu deaktivieren, falls dies nicht benötigt wird oder als störend empfunden wird. Es soll auch möglich sein, die ausgewählten Aktionen für die nächsten x Minuten auszuführen, damit der Nutzer in dieser Zeit nicht weiter gestört wird (siehe Abbildung 2).

Es ist dadurch möglich dem Nutzer ein System zur aktiven Unterstützung bei der Endbenutzer-Entwicklung anzubieten. Durch die zentrale Logik lässt sich dieses Konzept in die orchestrierte Kategorie einordnen. Die Widgets können ohne die zentrale JavaScript-Instanz nicht selbstständig miteinander kommunizieren. Ist das System einmal eingelernt, so ist eine automatisierte, indirekte Kommunikation zwischen den Widgets möglich – der Informationsfluss wird jedoch zentral vom JS-Helper gesteuert.

Abbildung 1 zeigt beispielhaft ein Wireframe eines Mashups. Der rote Rahmen soll dabei den eingebetteten JS-Helper darstellen. Durch eine Eingabe im Such-Widget oben links (Abbildung 1) soll eine vom Benutzer definierte Aktion ausgeführt werden. Bei der ersten Eingabe und dem anschließenden Senden der Nachricht mittels der Submit-Methode des Formulars kann sich der JS-Helper melden und Nachfragen, welche Aktion ausgeführt werden soll. Das Problem hierbei ist nicht die Herkunft der Daten zu bestimmen, sondern diese zum richtigen Ziel zu leiten und dort die richtige Aktion auszuführen. Dem Benutzer muss daher eine intuitive Eingabemöglichkeit gegeben werden, dies auf einfache Art und Weise zu bewerkstelligen. Abbildung 2 zeigt ein Wireframe einer möglichen Implementierung eines dialogbasierten Unterstützungs-systems.

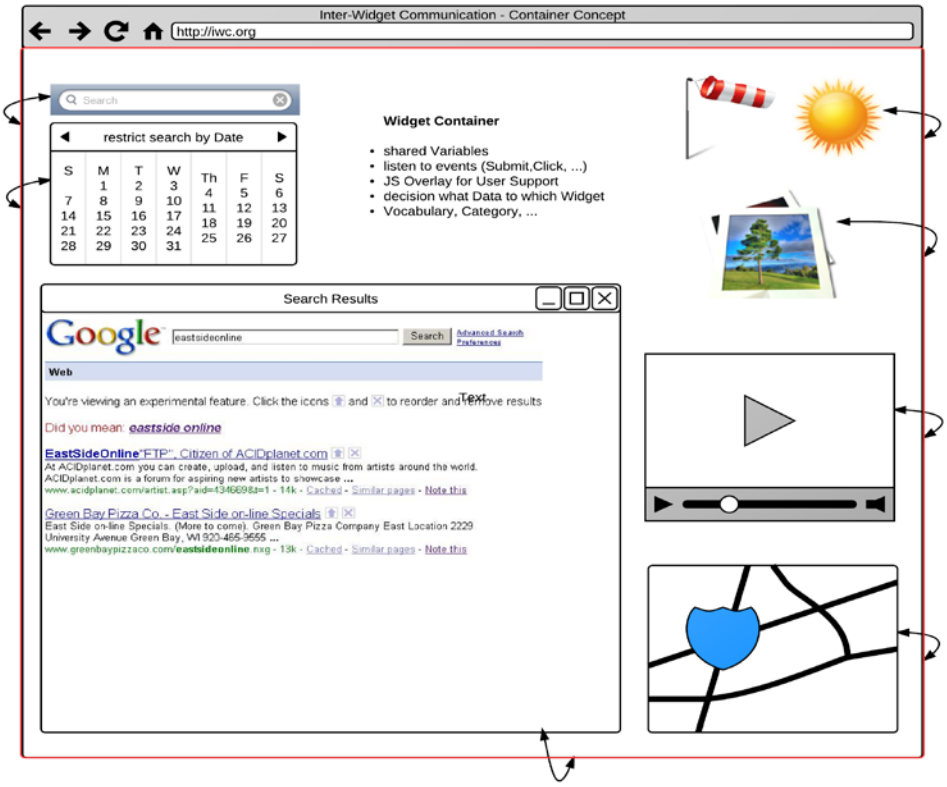

Abbildung 1: Wireframe des Containerkonzepts

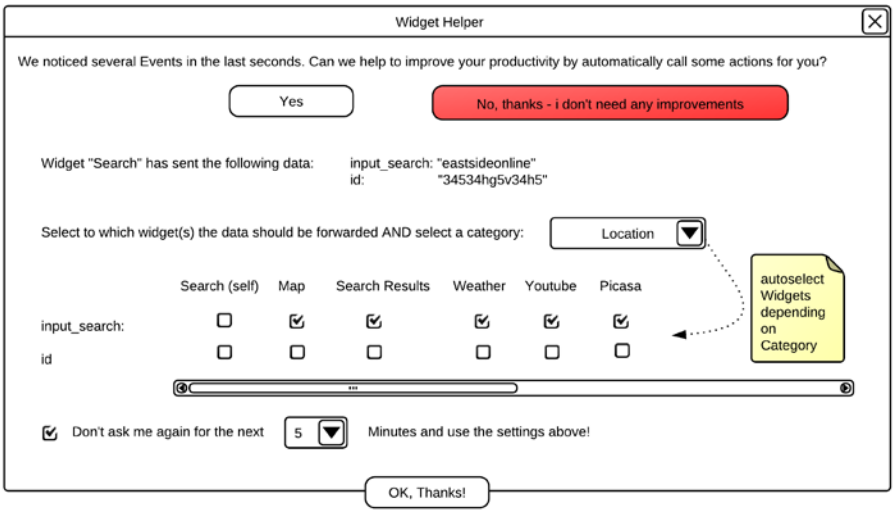

Abbildung 2: Dialogbasiertes JavaScript-Overlay des Unterstützungssystems

Eine Erweiterung des vorgestellten Ansatzes stellt die Identifikation von Quell- und Ziel-Widgets mittels Drag & Drop dar. Für den Nutzer ist dies intuitiver und damit einfacher zu handhaben. Entweder zieht der Nutzer die Daten direkt ins gewünschte Widget, indem er diese markiert und an die richtige Stelle im Ziel-Widget ablegt oder es wird mittels JavaScript ein Overlay angezeigt, wo der Nutzer von der Quelle zum Ziel einen Pfeil zieht und damit das Ziel eindeutig bestimmt. Damit der Inhalt im Ziel-Widget auch aktualisiert wird, muss bei den verschiedenen Aktionen jeweils die richtige Methode aufgerufen werden. In den meisten Fällen wird es eine Submit-Methode sein, es kann jedoch auch komplexere Ausmaße annehmen. Bei Änderungen von Radio- oder Checkboxen wird ggf. eine erweiterte dialogbasierte Methode nötig sein, da dies meist nicht sinnvoll via Drag & Drop abzubilden ist.

Um die nötigen Benutzereingaben weiter zu reduzieren, kann der JS-Helper beobachten welche Daten der Nutzer in welchem Widget eingibt und welche Aktionen ausgeführt werden. Gibt der Nutzer bspw. den Suchbegriff „eastsideonline“ im Such-Widget ein und kurz drauf wird der Begriff „eastsideonline“ auch im Wetter-Widget und im Map-Widget registriert, so kann der Nutzer, bspw. durch einen Balloon-Tipp, gefragt werden, ob die letzten Aktionen seit der ersten Eingabe des Begriffs in Zukunft automatisch ausgeführt werden sollen.

3.2 Helper-Widget-Konzept

Ist keine Möglichkeit zur Erweiterung der Plattform bzw. des Containers gegeben, so ist das gerade beschriebene Containerkonzept nicht anwendbar. Das Helper-Widget-Konzept basiert hingegen auf dem üblichen Publish-Subscribe (pubsub)-Verfahren. Der Container bzw. die Plattform muss dafür die Unterstützung des pubsub-Verfahrens anbieten. Anders als beim ersten Konzept müssen nun die Widgets durch das Einbinden

einer JavaScript-Datei (JS-Helper) ergänzt werden. Dieser JS-Helper sorgt dafür, dass sich ein Widget mit einem eindeutigen Topic am pubsub Hub anmeldet und alle Events und Daten des Widgets über dieses Topic sendet. Als Kodierung der zu sendenden Daten ist JSON zu nutzen. Für dieses Konzept ist es wichtig, dass es keine Widgets mit dem gleichen Topic gibt. Alle Widgets müssen zwingend unterschiedliche Topic-Namen besitzen. Dies ist für ein pubsub-Verfahren unüblich, aber für diesen Verwendungszweck notwendig.

Kernkomponente bei diesem Konzept ist ein Helper-Widget. Dies ist ein Widget, welches alle Topics abhört (subscribe: *), ein Unterstützungssystem anbietet und ähnlich dem JS-Helper aus dem ersten Konzept agiert. Durch das Abhören aller Topics bekommt das Helper-Widget alle Informationen der anderen Widgets und kann zentral entscheiden, an welche Topics die eingehenden Nachrichten weitergeleitet werden sollen. Auch hier ist wieder eine Lernphase notwendig bis eine automatisierte Weiterleitung der Nachrichten möglich ist. Da dieses Konzept Techniken der choreografischen Kategorie (pubsub-Methodik) und eine zentrale Komponente für die Weiterleitung der Nachrichten nutzt, ist dieses Konzept in die Kategorie der hybriden Mashups einzuordnen.

Abbildung 3 zeigt ein mögliches Mashup mit einem Helper-Widget. Am Rand jeden Widgets sind beispielhaft die Topic-Namen notiert. Das Helper-Widget empfängt bspw. eine Nachricht zu dem Topic „org.iwc.search“ vom Such-Widget. Es schaut in seinem Datenspeicher, ob bereits eine Regel dafür vorhanden ist und führt diese aus, falls dem so ist. Wenn keine Regel vorhanden ist, wird ein Dialogsystem angeboten, damit der Nutzer entscheiden kann, an welche registrierten Kanäle die Informationen gesendet werden sollen. Abbildung 4 zeigt einen möglichen Unterstützungsdialog.

Auch hier ist es notwendig, dass der Nutzer die Möglichkeit hat das System einfach zu deaktivieren. Durch die lose Kopplung beim pubsub-Verfahren, kann entweder das Widget aus dem Mashup entfernt werden oder es existiert im Widget selbst eine Funktion zur Deaktivierung (siehe Abbildung 3). Das Widget sollte außerdem verschiedene Ansichten haben, damit der Nutzer nicht zu viel Platz in seinem Mashup dafür bereitstellen muss. Denkbar ist eine Detailansicht mit den vergangenen Aktionen wie in Abbildung 3 zu sehen und eine normale Ansicht, wo nur ein Ein- bzw. Ausschalter zu sehen ist. Im Beispiel ist die Detailansicht der bereits ausgeführten Aktionen von unten nach oben zu lesen. Wie in Abbildung 4 zu sehen ist, ist auch eine Funktion zu integrieren, wo das System in den nächsten x Minuten die gleiche Aktion ausführen soll. Die Dialoge sollen zudem die Topic-Namen nutzerfreundlich darstellen. Dazu kann bspw. der umgekehrte Domänenname „org.iwc“ bei der Darstellung entfernt werden. Die gespeicherten Aktionen sollen ebenso bei der nächsten Verwendung des Mashups zur Verfügung stehen. Dies kann auch hier durch Cookies oder ein Loginsystem realisiert werden.

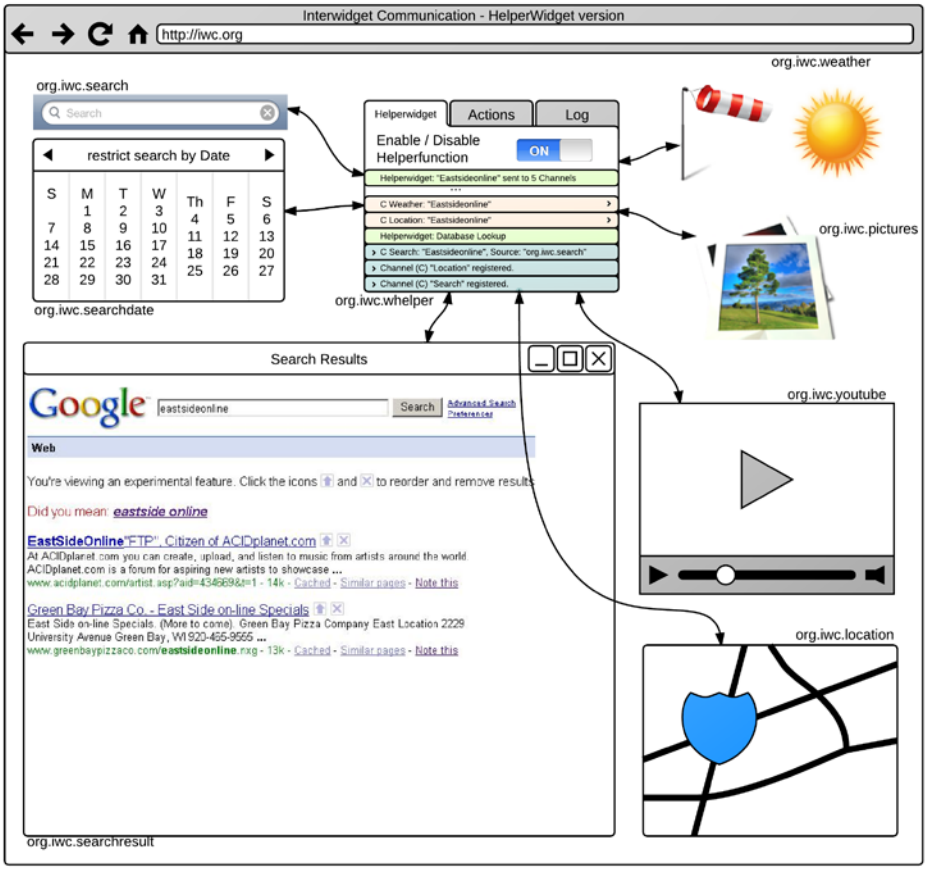

Abbildung 3: Wireframe eines Mashups mit einem Helper-Widget

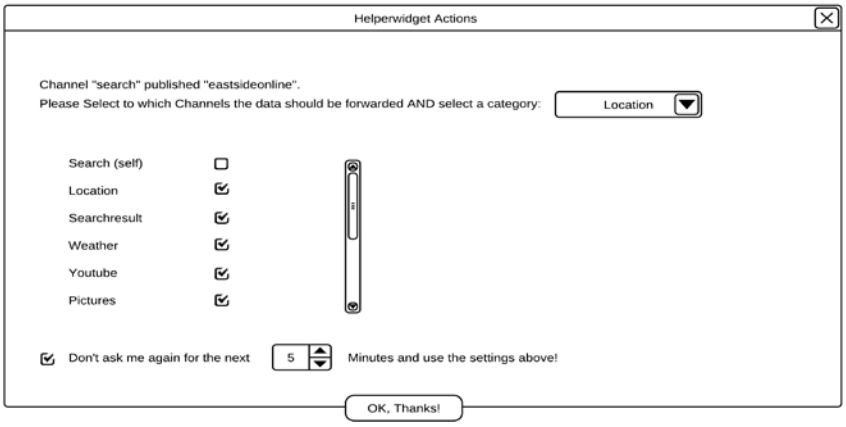

Abbildung 4: Unterstützungsdialog für das Helper-Widget

Um die Anzahl der nötigen Benutzereingaben zu minimieren, kann mit diesem Konzept auch ein passiveres Verhalten realisiert werden. Gibt der Nutzer in zwei Widgets die gleichen Daten ein, so kann dies auch vom Helper-Widget registriert werden und diese Zuordnung entweder stillschweigend übernommen werden oder es wird, wie in Abbildung 5 zu sehen, der Nutzer gefragt. Für manche Nutzer ist diese Variante besser geeignet als für andere, daher sollte in den Optionen des Helper-Widgets das Umschalten zwischen diesen Funktionsweisen möglich sein. Drag & Drop Operationen sind ebenso durch das Helper-Widget feststellbar. Dies wird durch die jeweiligen Events in den anderen Widgets und dem anschließenden Senden dieser Informationen via pubsub zum Helper-Widget realisiert. Damit können die Drag & Drop Aktionen auf die gleiche Weise behandelt werden (Abbildung 5).

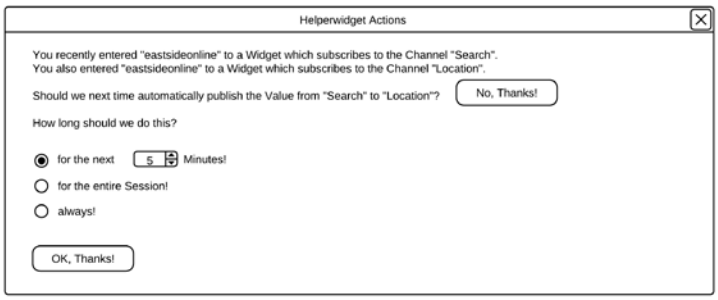

Abbildung 5: Unterstützungsdialog für das Helper-Widget bei doppelter Eingabe

Der Vorteil des Helper-Widget-Konzepts liegt bei der Nutzung vorhandener, etablierter Techniken. Der OpenAjax Hub bietet eine sichere Möglichkeit des Informationsaustausches innerhalb eines Browserfensters mit verschiedenen iframe- oder CSS-Widgets. Die Sicherheit des Gesamtsystems kann damit von vornherein auf einem hohen Niveau stattfinden. Ein Nachteil ist ganz klar die nötige Anpassung der Widgets durch das Laden der JavaScript-Erweiterung.

3.3 Mischformen und Cross-Browser / Intra-Widget Kommunikation

Je nach Anwendungsfall sind auch Mischformen der beiden genannten Konzepte möglich. Hat man bspw. die Möglichkeit den Container beliebig zu erweitern und es wird das pubsub-Verfahren genutzt, so können dadurch ggf. positive Ergebnisse erzielt werden. Man muss die Widgets unter Umständen nicht anpassen und gleichzeitig nutzt man ein bereits bewährtes pubsub-Verfahren.

Bietet eine Umgebung keine Möglichkeit der Anpassung des Containers und auch keine Funktionalität des pubsub-Verfahrens, kann über eine serverseitige Verteilung der Daten nachgedacht werden. Die Widgets müssten dann allerdings so erweitert werden, dass diese ihre Daten zu einem festgelegten Server senden. Dies ist wie im Helper-Widget-Konzept durch das Laden einer JavaScript-Erweiterung möglich. Dienste wie Firebase[8]

[8] http://www.firebase.com/

oder Pusher[9] bieten u. a. eine Möglichkeit die Daten auf schnellem Wege via Websockets ohne Long-Polling [11] an ihr Ziel zu leiten. Durch die Verteilung der Daten über einen Server sind alle genannten Bereiche der Widget-zu-Widget Kommunikation abdeckbar. So kann die IWC gleichermaßen wie die Intra-Widget und Cross-Browser Kommunikation realisiert werden. Im Allgemeinen müssen dafür allerdings die Widgets angepasst werden.

Gelingt es jedoch bspw., dass ein Container für jedes Widget standardmäßig eine Option für die Angabe eines Servers in Form einer URL hinzufügt und bei jedem Widget eine global definierte Erweiterung injiziert (ähnlich dem JS-Helper beim Helper-Widget-Konzept), so kann der Datenaustausch über einen zentralen Server erfolgen ohne dass die originalen Widgets angepasst werden müssen. Der Nutzer kann in diesem Fall bei jedem Widget einen Zielserver über die Widget-Optionen definieren. Es bietet sich dafür ein serverseitiges pubsub-Verfahren an.

4 Fazit und Ausblick

In dieser Arbeit wurden mehrere Ansätze zur Unterstützung der Endbenutzer-Entwicklung mittels UI-Mashups und IWC vorgestellt. Dabei wurden verschiedene Fälle zur Anpassung des Containers, der Widgets und ebenso Mischformen betrachtet. Es wurden mehrere konzeptuelle Ideen entworfen, welche für die Ansätze weiterer Arbeiten dienen können. Es fehlt eine prototypische Implementierung dieser Ansätze um deren Funktionstüchtigkeit im Alltag zu erproben. Es ist daher geplant, eine Implementierung auf Basis von Apache Rave, Apache Wookie und dem OpenAjax Hub zu realisieren.

Die vorgestellten Konzepte zielen auf eine benutzerfreundliche Lösung damit ein Endanwender die Widget-zu-Widget Kommunikation einfacher in ein von ihm zusammengestelltes Mashup integrieren kann. Es wird dafür ein aktives Hilfesystem angeboten. Ist die Integration der IWC erst einmal erfolgt, kann der Nutzer die Produktivität aufgrund der weniger anfallenden Benutzereingaben erhöhen. Die manuellen Zustandssynchronisierungen zwischen den Widgets werden auf ein Minimum reduziert.

Kommerzielle Dienste wie bspw. Firebase.com oder Pusher.com ermöglichen eine schnelle Übertragung von Informationen über einen Server. Diese Möglichkeit kann auch für die Widget-zu-Widget Kommunikation genutzt werden. Da diese Dienste jedoch kommerziell vermarktet werden, ist die Zukunft der Dienste ungewiss und die Sicherheit der abgelegten Daten ist auch zu hinterfragen. Um die volle Kontrolle des Systems zu behalten, ist es notwendig einen eigenen Server aufsetzen zu können. Sinnvoll wäre bspw. ein pubsub-Server mit einer Websocket-Anbindung um unnötige Latenz zu vermeiden. Um eine sichere Umgebung zu gewährleisten, muss bei den gezeigten Konzepten die Sicherheit weiter analysiert werden.

[9] http://pusher.com/

5 Literaturverzeichnis

[1] S. Wilson, F. Daniel, und U. Jugel, “Orchestrated User Interface Mashups Using W3C Widgets,” *Current Trends in Web Engineering*, S. 49-61, 2012.

[2] “Mashup Definition.” [Online]. Verfügbar: http://www.techterms.com/definition/mashup. [Letzter Zugriff: 24.05.2012].

[3] “Widget Packaging and XML Configuration.” [Online]. Verfügbar: http://www.w3.org/TR/widgets/#definitions. [Letzter Zugriff: 24.05.2012].

[4] I. Zuzak, M. Ivankovic, und I. Budiselic, “A Classification Framework for Web Browser Cross-Context Communication,” *Arxiv preprint arXiv:1108.4770*, 2011.

[5] OpenAjax Alliance, “OpenAjax Hub 2.0 Specification,” 2009. [Online]. Verfügbar: http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification. [Letzter Zugriff: 01.06.2012].

[6] “OpenSocial Core Gadget Specification 2.0.1.” [Online]. Verfügbar: http://opensocial-resources.googlecode.com/svn/spec/2.0.1/Core-Gadget.xml#interGadgetEventing. [Letzter Zugriff: 24.05.2012].

[7] Google Inc, “Removing social functionality from iGoogle gadgets - iGoogle — Google Developers,” 2012. [Online]. Verfügbar: https://developers.google.com/igoogle/docs/removing_social. [Letzter Zugriff: 01.06.2012].

[8] W3C, “Widget Packaging and XML Configuration,” 2011. [Online]. Verfügbar: http://www.w3.org/TR/widgets/. [Letzter Zugriff: 01.06.2012].

[9] S. Sire, M. Paquier, und A. Vagner, “A Messaging API for Inter-Widgets Communication,” *Proceedings of the 18th international conference on World wide web*, S. 1115-1116, 2009.

[10] O. Chudnovskyy, S. Müller, und M. Gaedke, “Extending Web Standards-based Widgets towards Inter-Widget Communications (to be published),” *ComposableWeb 2012*, 2012.

[11] I. E. T. Force, “RFC 6202,” 2011. [Online]. Verfügbar: http://tools.ietf.org/pdf/rfc6202.pdf. [Letzter Zugriff: 01.06.2012].