From Choreographed to Hybrid User Interface Mashups: A Generic Transformation Approach

Alexey Tschudnowsky¹, Stefan Pietschmann², Matthias Niederhausen²,
Michael Hertel¹, and Martin Gaedke¹

¹ Technische Universität Chemnitz, Germany

{alexey.tschudnowsky,michael.hertel,gaedke}@informatik.tu-chemnitz.de

² T-Systems MMS, Germany

{stefan.pietschmann,matthias.niederhausen}@tu-dresden.de

Abstract. Inter-widget communication (IWC) becomes an increasingly important topic in the field of user interface mashups. Recent research has focused on so-called choreographed IWC approaches that enable self-organization of the aggregated components based on their messaging capabilities. Though a manual configuration of communication paths is not required anymore, such solutions bear several problems related to awareness and control of the emerging message flow. This paper presents a systematic approach to tackle these problems in the context of *hybrid* user interface mashups. We show how users can be made aware of the emerged IWC configuration and how they can adjust it to their needs. A reference architecture for development of hybrid mashup platforms, is derived and one implementation based on the publish-subscribe choreography model is given. We report on the results of a first user study and outline directions for the future research.

Keywords: inter-widget communication, user interface mashup, widgets, end-user development.

1 Introduction

User interface (UI) mashups have become a popular approach for end-user development. Based on autonomous but cooperative visual components called widgets, they promise to significantly lower the barrier for Web application development [2,19,10]. The development process of UI mashups usually implies three steps: First, finding appropriate widgets for composition; Second, placement and configuration of widgets on a common canvas; and finally, configuration of the cooperative behaviour by means of inter-widget communication (IWC). IWC hereby refers to the process of exchanging data between widgets and can be used to synchronize internal states of the aggregated components.

The IWC behaviour of a mashup can be defined either explicitly by a mashup designer (*orchestrated mashups*), emerge from the capabilities of the integrated components (*choreographed mashups*) or be defined by a combination of both (*hybrid mashups*) [22]. While orchestrated approaches aim at providing flexibility during mashup development by enabling designers to define the desired data

flow manually, choreographed and hybrid solutions focus on keeping the development process lean and fast. In choreographed solutions, widgets “decide” autonomously on how to communicate and with whom. Hybrid mashups behave as choreographed ones, but provide additional means to restrict the emerging communication. EDYRA [19], DashMash [3] or EzWeb [13] are some approaches that exemplify orchestrated mashups. Using the “wiring” metaphor, they enable mashup designers to connect “inputs” and “outputs” of components and, thus, specify data flow in a mashup. The target group of such platforms are skilled users and hobby developers, who are experienced with the concepts of operations, input/output parameters and data structures. OMELETTE [4], ROLE [10] and Open Application [8] projects follow the choreographed approach. Widgets communicate without a prior configuration - using the publish-subscribe messaging pattern each widget decides autonomously on which messages to send and which messages to receive. Chrooma+ [12] is an example of a hybrid platform. While widgets publish and subscribe for messages on their own, a mashup designer is still able to “isolate” one or more widgets from their environment. The target group of choreographed and hybrid platforms are end-users, who have little to no programming skills but are experts in their corresponding business domains.

Though choreographed and hybrid mashups are considered to be more “end-user-friendly” [8], they also pose some challenges with regard to awareness and control of what is happening in a mashup [21,8]. The major awareness problem caused by implicitly defined IWC is that users do not know which pairs of widgets *could* and which actually *do* communicate. Users have to learn the data and control flows as they use and explore the mashup. While in general this may merely frustrate users, such “exploratory” interaction can also accidentally affect live data, causing undesired side effects. The major control problem is that, being defined implicitly and not as first-level concepts, communication paths cannot be blocked, modified or added directly by end-users. Possible reasons for intervention are, e.g., untrusted widgets or unexpected or undesired state synchronisations. A detailed analysis of these and other problems related to awareness and control in UI mashups can be found in our prior work in [5].

This paper presents an approach for systematic development of hybrid mashup platforms with IWC awareness and control in mind. The goal is to support end-user development of UI mashups by combining advantages of choreographed and orchestrated mashup platforms - self-emerging IWC with flexible visualization and tailoring facilities. In summary, the contributions of this paper are as follows:

- *A generic IWC model including corresponding visualization and tailoring mechanisms.* The model and mechanisms are used to communicate IWC behaviour to end-users and facilitate its configuration.
- *A reference architecture for hybrid mashup platforms.* The architecture enables systematic development of hybrid mashup platforms - both from scratch and by extension of existing choreographed ones.
- *Evaluation of the proposed concepts with end-users.* Experiments with 27 end-users assessed the efficiency and usability of the proposed mechanisms.

The rest of the paper is structured as follows. Section 2 gives a background on choreographed UI mashups and presents the steps required for systematic development of hybrid mashup platforms. Section 3 demonstrates one implementation of the proposed architecture for the publish-subscribe choreography strategy. An evaluation of the proposed awareness and control facilities is given in section 4. Finally, section 6 concludes the paper and derives directions for future research.

2 From Choreographed to Hybrid UI Mashups

The section presents a systematic approach to build hybrid mashup platforms. As the working principle of hybrid mashups is close to choreographed ones, the idea is to leverage existing choreographed platforms and to extend them with the missing visualization and tailoring functionality. The approach can also be used to build hybrid platforms from scratch.

In the following, different types of choreographed mashups with regard to utilized IWC models are presented. Afterwards, a unified communication model for visualization and tailoring of data flow is described. Based on the model, several awareness and control mechanisms are proposed. Finally, a reference architecture to support these mechanisms is given.

2.1 Choreographed UI Mashups

Choreographed UI mashups do not require mashup designers to specify data flow in a mashup. Instead, communication emerges in a self-organizing fashion depending on the messaging capabilities of widgets. Technically, different IWC strategies exist to enable "self-organization" of autonomous but cooperative components: message passing, publish-subscribe, remote procedure calls (RPC) and shared memory[24]. *Message passing* considers widgets as senders and recipients of structured application-specific data. Delivery can take place in uni-, multi- or broadcast fashion. *RPC* solutions enable widgets to offer operations, which can be invoked by others in synchronous or asynchronous way. Discovery of available operations happens either at widget design-time or at run-time, e.g., by means of centralized widget registries. In *publish-subscribe* systems, widgets emit and receive messages on different channels, also called "topics". The decision, which channels to publish or subscribe on, is met by widgets autonomously without intervention of a mashup designer. Finally, *shared memory* solutions enable widgets to read and to write to a common data space and, thus, autonomously exchange data among each other. Independently of the concrete communication model, widgets can communicate either directly or by means of the platform middleware. Platform-mediated communication enables loose coupling of aggregated components as well as additional services such as traffic monitoring and management. As to the authors' knowledge, all of the current UI mashups make use of platform middleware to implement IWC. The presented strategies

offer techniques for implementation of self-organizing widget compositions. User awareness and control are out of their scope and thus require additional engineering on the side of the mashup development/execution platform.

2.2 Communication Model

The goal of the unified communication model presented here is to provide a common data structure for visualization and control mechanisms. The assumption for the definition of the model is that – from an end-users’ point of view – widgets communicate in pairs by means of unidirectional message transfers. In terms of a concrete choreography model, messages have different semantics, e. g., invocation of a remote procedure or publication/subscription to some topic. Regardless of the model, the user-perceived result is that one widget receives data from another one. These considerations build a basis for the unified communication model described by the following data structure:

The unified communication model M is a graph $G = (V, E)$ with

- $V = \{v | v = (id, s)\}$ set of vertices with identifier id and state $s \in \{ENABLED, ISOLATED\}$. Each vertex corresponds to exactly one widget in a mashup.
- $E = \{e | e = (v_1, v_2, s, t)\}$ set of edges corresponding to possible communication paths between widgets corresponding to $v_1, v_2 \in V$ with state $s \in \{ENABLED, BLOCKED\}$ and with label t .

For all of the presented choreography models it is possible to define an algorithm which yields a unified communication model M . Section 3.2 presents one possible algorithm for publish-subscribe-based choreography models.

A data flow restricted by the unified model M takes place as follows:

- A widget corresponding to the vertex v is allowed to emit or receive messages only if the state s of the vertex v is *ENABLED*.
- A message m from a widget corresponding to v_1 is allowed to be delivered to a widget corresponding to v_2 only if $\exists e \in E : e = (v_1, v_2, ENABLED, t)$.
- The data flow takes place according to the utilized choreography strategy if none of the above restrictions apply.

2.3 Visualization and Tailoring Facilities

The following visualization and tailoring facilities are proposed to make mashup designers aware of the data flow in a UI mashup and to enable them to adjust it:

- States $s \in \{ENABLED, ISOLATED\}$ of vertices are visualized using borders of different color and type around the corresponding widgets.
- Potential communication paths $e = (v_1, v_2, s, t) \in E$ are visualized using arrows between widgets corresponding to v_1 and v_2 . The arrow style indicates the state of the communication path $s \in \{ENABLED, BLOCKED\}$. Annotation t is displayed above the corresponding arrow to provide additional information on the communication path.

- Flashing icons on widget borders show which widgets corresponding to v_1 and v_2 are currently communicating along a communication path $e = (v_1, v_2, s, t) \in E$.
- For every vertex v , visualization of its state $s \in \{ENABLED, ISOLATED\}$ and in-/outgoing edges $e = (v, *) \in E$ can be turned on or off to avoid cognitive overload in case of strong connectivity.
- For every vertex v , its state $s \in \{ENABLED, ISOLATED\}$ can be toggled using corresponding user interface controls.
- For each edge $e = (v_1, v_2)$, its state $s \in \{ENABLED, BLOCKED\}$ can be toggled by clicking on the corresponding arrows.

2.4 Reference Architecture

The reference architecture (cf. Figure 1) acts as a blueprint for the development of awareness- and control-enabled hybrid mashup platforms. Some of the platform components (such as the *Awareness and Control Module* and the *Communication Model*) are independent of a chosen choreography model, whereas others (*Widget and Mashup Descriptors*, *Model Importer*, *Model Exporter* and *Message Broker*) are choreography-model-specific.

Fig. 1. Reference architecture for implementation of hybrid UI mashups. * - implementation can be shared between platforms with different choreography strategies.

The components of the reference architecture provide resources and services required for UI mashup development. *Widget Descriptors* describe IWC capabilities and default configuration parameters of installed widgets. Mashup layout, aggregated widget instances, user preferences and IWC configuration are

specified by the *Mashup Descriptor*. All artefacts are dependent on the selected choreography model, but can be used by the *Model Importer* to derive a unified *Communication Model*. The *Awareness and Control Module* displays the model in the composition canvas according to the rules described in 2.3. It is also responsible for updating the *Communication Model* upon changes triggered by users, e.g., changing state of an edge upon clicks on the corresponding arrow. The *Message Broker* is the platform's communication middleware. Its goal is, first, to provide messaging functionality according to the rules of the underlying choreography approach and, second, to assure that restrictions of the unified *Communication Model* such as isolated widgets or blocked communication paths are respected. The *Awareness and Control Module* gets notified about activities within the *Message Broker* and displays activated or blocked communication paths to mashup users.

In the following, we present one example implementation of the reference architecture. It reuses an existing publish-subscribe-based UI mashup platform and extends it towards the missing awareness and control functionality. We selected a platform based on the publish-subscribe strategy because of the wide use of this approach in current choreography platforms.

3 Hybrid Mashups Based on Publish-Subscribe Choreography Model

Publish-subscribe-based IWC is a part of many choreographed UI mashup platforms [4,12,10]. Frameworks like OpenAjaxHub¹ or AmplifyJS² simplify the integration of the corresponding infrastructure into Web-based applications. The underlying communication strategy is, however, always the same and can be formalized as follows:

Let $m = (W, T)$ be a widget mashup with

- Reference ontology $T = \{t_n : t_n = (name_n, TYPE_n)\}$ being a set of concepts and associated message types $TYPE_n = \{value_n\}$
- Widget set $W = \{w_j : w_j = (id_j, PUB_j, SUB_j)\}$ with unique identifier id_j , set of publications $PUB_j = \{p_{jl} : p_{jl} \in T\}$ and set of subscriptions $SUB_j = \{s_{jk} : s_{jk} \in T\}$

Let $a = (wsender, t, data)$ with $wsender \in W, t \in wsender.PUB, data \in t.TYPE$ be a message emitted by a widget $wsender$. The message a is delivered to all widgets $w_i \in W : t \in w_i.SUB$.

The following implementation of a publish-subscribe-based hybrid mashup platform is based on two open-source projects - Apache Wookie³ and Apache Rave⁴. Apache Wookie is a widget container for hosting W3C widgets⁵, which are

¹ http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification

² <http://amplifyjs.com>

³ <http://wookie.apache.org>

⁴ <http://rave.apache.org>

⁵ <http://www.w3.org/TR/widgets>

stand-alone Web applications packaged for distribution on the Web. Apache Rave is a widget mashup environment, which enables aggregation of both W3C and OpenSocial⁶ widgets on one canvas and provides a publish-subscribe messaging infrastructure for communication between the widgets. In the following, those parts of the projects are described in detail, which were extended towards the reference architecture.

3.1 Widget Descriptors

According to the W3C specification, widget metadata contains only basic information on the packaged application such as title, description, author, license etc. Currently there is no standard way to describe widget communication capabilities. However, this is essential for the proposed awareness and control mechanisms (cf. section 2.2). Therefore, an extension of the W3C metadata file is proposed to describe publications and subscriptions of widgets together with used topics. Listing 1.1 shows the proposed extension for a fictitious *Contacts* widget, which maintains a list of structured contact entries. An incoming phone number (topic *http://example.org/phoneNumber*) causes the widget to filter the list towards a contact with the passed number. If a contact is selected by user, its complete data is published on the *http://example.org/contact* topic. The schema of the involved messages is given in the *oa:topics* section. Both Apache Wookie and Rave were extended to support the extended widget metadata.

Listing 1.1. Extension of the W3C metadata file

```
<feature name="http://www.openajax.org/hub"
xmlns:oa="http://www.openajax.org/hub">

<!-- Declaration of topics -->
<oa:topics>
  <oa:topic oa:name="http://example.org/contact">
    <oa:schema oa:schemaType="JSON">
      <![CDATA[
        { "description": "A person",
          "type": "object",
          "properties": {
            "name": { "type": "string" },
            "age" : {
              "type": "integer",
              "maximum": 125 }
          }
        }
      ]]>
    </oa:schema>
  </oa:topic>

  <oa:topic oa:name="http://example.org/phoneNumber">
```

⁶ https://developers.google.com/gadgets/docs/dev_guide?csw=1

```

    <oa:schema oa:schemaType="XML" oa:simpleType="xs:string" />
  </oa:topic>

  <oa:topic ...>
</oa:topics>

<!-- Declaration of publications -->
<oa:publications>

<oa:publication oa:topicRef="http://example.org/contact" />
  <oa:publication ...>
</oa:publications>

<!-- Declaration of subscriptions -->
<oa:subscriptions>
  <oa:subscription oa:topicRef="http://example.org/phoneNumber" />

  <oa:subscription ...>
</oa:subscriptions>

</feature>

```

3.2 Model Importer

The module has been added to Apache Rave and is responsible for construction of the *Communication Model* based on *Widget Descriptors* and *Mashup Descriptors*. The construction of the model for publish-subscribe-based UI mashups is specified by the Algorithm 1.

Algorithm 1. Creating a Unified Communication Model for Publish-Subscribe-based UI Mashups

Input : Publish-subscribe-based UI mashup $m = (W, T)$ as defined above

Output: Communication model $G = (V, E)$ as defined in section 2.2

1. Set $V = \emptyset$, $E = \emptyset$
 2. For each widget $w_i \in W$, create a new vertex v_i and set $V = V \cup v_i$
 3. For each pair of widgets (w_i, w_j) and for each $p_{ik} \in w_i.PUB : p_{ik} \in w_j.SUB$, create a new edge $e_{ik} = (v_i, v_j, ENABLED, p_{ik})$ and set $E = E \cup e_{ik}$
 4. Return $G = (V, E)$
-

The resulting communication model reflects potential message flows in a publish-subscribe-based UI mashup. By default, all communication paths and widgets are in the *ENABLED* state.

3.3 Awareness and Control Module

The *Awareness and Control Module* weaves the *Communication Model* into the composition canvas and updates it, based on user actions such as widget isolations or path blockades. It also highlights communicating partners according to notifications from the *Message Broker*. Visualization of the model takes place as proposed in Section 2.3 and is implemented using the jsPlumb⁷ JavaScript drawing library (cf. Figure 2).

Fig. 2. Awareness and Control Mechanisms integrated into Apache Rave

3.4 Message Broker

The *Message Broker* is an existing component in the Apache Rave platform and is implemented using the OpenAjaxHub framework. It is responsible for routing messages between widgets according to the publish-subscribe strategy as specified above. The component has been extended to take the *Communication Model* into account while routing messages. The routing algorithm was refined as follows: a message $a = (wsender, t, data)$ from widget $wsender$ is allowed to be delivered to $\forall w_i \in W : t \in w_i.SUB$ if and only if $v_{wsender}.s = ENABLED \wedge v_{w_i}.s = ENABLED \wedge e.s = ENABLED : e = (v_{wsender}, v_{w_i})$.

⁷ <http://jsplumbtoolkit.com/>

3.5 Model Exporter

The *Model Exporter* component enables serialization of the current communication model and its integration into the platform-specific *Mashup Descriptor*. Apache Rave makes use of the OMDL⁸ format to describe the configuration of UI mashups. However, the current specification doesn't provide any means to include IWC configuration into the mashup specification. Thus, we propose to extend OMDL documents with missing information on the *Communication Model*. An example of an extended *Mashup Descriptor* in OMDL format is given in Listing 1.2. The goal of the described mashup is to provide aggregated information on emergency incidents during natural disasters such as in case of flood. The mashup aggregates four widgets and defines IWC restrictions for the current composition. One of the widgets (flood graph) is completely isolated. Such configuration is useful to fix view of a widget and to avoid refreshes caused by changes others. Communication between the map widget and the contacts widget is forbidden only for the topic *http://example.org/phoneNumber*. It results in the behavior, that no marker selection in the map will refresh the contacts widget, while activities in other widgets may do.

Listing 1.2. OMDL mashup description and proposed extension

```
<workspace xmlns="http://omdl.org/"
  xmlns:oa="http://www.openajax.org/hub">
  <identifier>http://example.org/mashup/379</identifier>
  <title>Dresden Flood</title>
  <description></description>
  <creator>Alexey</creator>
  <date>2013-06-18T14:39:58+0200</date>
  <layout>THREE COLUMNS</layout>

  <app id="http://example.org/incidentMap-1">
    <link href="http://example.org/s/incidentsMap.wgt"
      type="application/widget" rel="source"/>
    <position>LEFT TOP</position>
  </app>

  <app id="http://example.org/contacts-1">
    <link href="http://example.org/s/contacts.wgt"
      type="application/widget" rel="source"/>
    <position>LEFT MIDDLE</position>
  </app>

  <app id="http://example.org/floodGraph-1">
    <link href="http://example.org/s/floodGraph.wgt"
      type="application/widget" rel="source"/>
```

⁸ <http://omdl.org>

```

    <position>RIGHT TOP</position>
  </app>

  <app id="http://example.org/floodGraph-2">
    <link href="http://example.org/s/floodGraph.wgt"
      type="application/widget" rel="source" />
    <position>RIGHT MIDDLE</position>
  </app>

  <!-- Proposed extension specifying mashup IWC behaviour -->

  <!-- Isolate a widget completely -->
  <oa:pubsub-restriction
    oa:source="*"
    oa:target="http://example.org/floodGraph-1"
    oa:topicRef="*" />
  <oa:pubsub-restriction
    oa:source="http://example.org/floodGraph-1"
    oa:target="*"
    oa:topicRef="*" />

  <!-- Forbid two widgets to communicate over the specified topic -->
  <oa:pubsub-restriction
    oa:source="http://example.org/incidentMap-1"
    oa:target="http://example.org/contacts-1"
    oa:topicRef="http://example.org/phoneNumber" />

</workspace>

```

This section presented an implementation of the reference architecture for the publish-subscribe choreography strategy. Application of the approach to other strategies requires to implement strategy-specific *Model Importer*, *Model Exporter* and *Message Broker* components. The *Awareness and Control Module* and *Communication Model*, however, can be reused across different platforms with minor adaptations.

4 Evaluation

To evaluate the presented approach in practice, we explored the following three hypotheses:

1. *Users solve tasks that require IWC faster if awareness and control mechanisms are used.* The intent was to check if the proposed mechanisms increase efficiency of end-users while working with UI mashups. The hypothesis was considered to be approved if the time needed to complete a given task with activated awareness and control facilities was lower than without them.

2. *Users find out easier, which widgets are connected, if IWC is visualized.* The intent was to check if the awareness mechanisms help users to spot, understand, and make use of the connections between widgets. The hypothesis was considered to be approved if average assessment on the ease of finding connections between widgets was higher with awareness facilities than without. The hypothesis was checked with two different types of widgets being employed: one group had mostly static content, while the other included animated changes. The rationale was that animated changes in widgets themselves might be sufficient on their own to identify communication partners, thereby making additional awareness mechanisms unnecessary.
3. *Users find the proposed control mechanisms easy to use.* The intent was to check if users felt comfortable with the proposed tailoring facilities. The approval condition was that more than 60% of participants agree or strongly agree that control facilities are easy to use.

To test the introduced hypotheses we applied the laboratory experiment methodology. Overall, 27 participants took part in the user study. Almost 90% of participants had no programming skills but were experts in the domain of marketing and telecommunication. The majority of users (74%) had an understanding of the term "widget", which was mostly related to mobile devices and the "Windows Vista Sidebar". Only 4 out of 27 users had ever configured a portal UI (mostly intranet portal) on their own, e.g., by repositioning of widgets and changing the colour scheme.

For each of the 27 participants, the evaluation procedure took about one hour and involved the following steps: Before the task execution, participants filled in a pre-evaluation questionnaire to judge their skill levels. Based on the results, they were evenly distributed over test and control groups. After that, users were given an introduction on the widget mashup platform, its purpose, concepts (mashups, widgets, etc.) and core functionalities. Following the introduction, users had the chance to explore and try out different aspects of the portal as they liked. After then, participants were asked to complete three tasks targeting different aspects of the system. The completion time was measured for each group and for each task. In a standardized post-questionnaire, users could express their subjective opinions on the introduced facilities. The study used a two-tailed t test, with a significance level of 95

Awareness Facilities. In the first experiment, users were asked to play a game on a dedicated mashup, called "Easter Egg Hunt" (cf. Figure 3, left). The goal of the game was to find the Easter egg in each of the nine visible widgets. Only one egg was visible at a time. Upon clicking it, the egg would disappear and appear in another widget. Thus, users quickly had to find the corresponding widget and the egg inside. For the test group, visualization of communicating widget pairs was enabled, so that outgoing and incoming data were displayed in the widget header. Thereby, the data flow could be perceived by users and, ideally, they could deduce where to look next. The control group accomplished the task without IWC visualization. The experiment was conducted in two different setups:

Fig. 3. Evaluation mashups for testing end-user efficiency with awareness (left) and control (right) facilities

for animated content (the egg would “fall” into place) and static content (the egg would just appear). The time required for completion of the task was measured.

Control Facilities. In the second experiment, participants had to solve a comparison task using two widgets (cf. Figure 3, right). One widget gave an overview of flat offers in a selected city and the other one showed details of the offer. The task was to find the cheapest flat in one city and then a comparable one in another city. While the control group used the default setting with one “overview” and one “detail” widget, the test group was provided with two “detail” widgets, one of which they could isolate from communication to simplify the comparison. Once isolation was enabled for a detail widget, it would “freeze”, so that new details could be loaded in the second widget and easily be compared to the first one. The time required for completion of the task was measured.

4.1 Results

Time measurements and evaluation of the post-questionnaire results yielded the following findings:

Hypotheses 1: End-User Efficiency. The results indicate a possible advantage of IWC visualization for widgets with mostly static content, i. e., whenever changes due to IWC are rather subtle as opposed to animations (cf. Figure 4, left). However, this difference is not statistically significant. When changes were animated by widgets, the average task completion times in test and control groups were roughly the same. Statistically, the test group was 18% slower than the control group (95% confidence), revealing a possible distraction of users due to the visualization.

This can be partly attributed to the overlapping of indicator flashes for fast users. The indicators were still flashing from their last interaction, when data was published by a new widget. This confused several users, who expected updated data, i. e., the egg, to appear in those widgets. The hypothesis for the awareness mechanisms is, thus, considered to be *not approved*.

Fig. 4. Impact of awareness (left) and control (right) facilities on end-user efficiency

The control mechanisms for IWC, namely the possibility to isolate widgets, gave users a slight benefit when solving the flat comparison task (cf. Figure 4, right). As the time advantage is not significant, this hypothesis is also considered to be *not approved*.

Hypotheses 2: Usability of the Awareness Mechanisms. According to user ratings (cf. Figure 5, left), IWC visualization does not help users in subtly suggesting “where to look next” and thus understanding which widget are communicating. For users in the test group, it made no difference whether changes in widgets were animated or not. In contrast, users from the control group obviously found it easier to “follow the egg” if the changes were animated, since those were easier to spot in their peripheral field of vision. Thus, for mostly static widgets IWC visualization seems to compensate the missing IWC indicator and to facilitate the recognition of communicating parties.

In the light of the above results, the hypothesis is considered to be *approved* for widgets with mostly static content and *not approved* for widgets with animated changes. Based on this fact, we can derive a guideline for widget developers to animate changes triggered by IWC in order to improve usability of the future mashups.

Fig. 5. Usability of the proposed awareness (left) and control (right) facilities

Hypotheses 3: Usability of the Control Mechanisms. IWC control mechanisms got very positive response. 64% of users found them easy to use (cf. Figure 5, right). The controls for this feature, namely the integration with the widget menu, were also rated positively. The hypothesis is considered to be *approved*.

In the post-questionnaire, the vast majority of the participants described the visualization and control facilities as helpful and easy to use. Users recommended making the IWC visualization optional and as such less subtle. One suggestion was to enable/disable visualization per widget by clicking the indicators directly. Furthermore, they suggested using a more noticeable colour scheme for indicating communication partners. It was proposed to investigate if a distinction between incoming/outgoing data is necessary to be visualized, or rather one indicator for taking part in data exchange is enough, e.g., a cogwheel. Finally, some participants suggested IWC controls, e.g., “Isolation”/”Pinning”, to be accessible from the widget header bar.

5 Related Work

The need for appropriate awareness and control facilities in choreographed user interface mashups has been recognized by a number of research projects and initiatives [8,21]. As for authors’ knowledge, the only hybrid mashup platform proposed at the moment is Chrooma+ [12], whereas only spare configuration of widget IWC behaviour is possible. In [21], Wilson proposes several ideas to tackle awareness and control challenges in UI mashups. Our work continues the research on this field and proposes actual solutions as well as corresponding architectural components.

The problem of visualizing and controlling interactions between autonomous entities is also tackled in other research areas. For example, in the field of self-organizing multi-agent systems, much research has been performed on the visualization of agent interactions. Typical strategies are, e.g., either to draw a graph of interactions in a whole system [1,6] or to highlight relationships between single agents and their environment.[14,17,15]. In [20], the authors propose to draw a causality graph to visualize message exchange in a multi-agent system. Though awareness of relationships in a multi-agent system can be increased with their approach, an a priori simulation and an event log are needed for the system to work. For mashups that employ cost-causing widgets (e.g., with telecommunication functionality), an a priori simulation might be undesirable.

An approach similar to the presented one is given in [16]. The authors propose a set of tools with different perspectives for monitoring and debugging of multi-agent systems. Relationships between agents and possible interactions are shown in the so-called ”society tool”. A requirement for the tool is that agents expose their partial knowledge about the outer world and communication capabilities to the tool, which then visualizes them in a graph fashion. Several controlling tools enable modification of agent states and configuration of their reactions on incoming messages. Though fine-grained exploration and adaptation of the system is possible, the target group of the approach are skilled developers.

In the field of natural programming environments, a common practice is to use natural language and question-answering games to explore a system. The WhyLine tool [11] applies natural language to enable unskilled developers to debug their algorithms. Using menus and pictograms of objects involved into an algorithm, developers can construct "why did" and "why did not" questions in order to explore system behaviour. A user study revealed that the participants were more efficient with this system than with traditional debugging tools. The approach is applied for explanation of static information and recorded event log. However, it doesn't foresee any means for visualization of active data transfer as it is required for IWC scenarios.

Similar research on control facilities in the context of loosely coupled communicating components can be found among visual programming tools. Lego-Mindstorms products⁹ based on LabVIEW¹⁰ enable unskilled developers to design their algorithms in a graphical way. Different boxes representing robot functionalities like move, rotate or stop can be connected with each other and controlled using loops or branches. The tool applies the "wiring" metaphor to connect inputs and outputs of the components, which makes the approach flexible and extensible. Many mashup platforms have adopted the "wiring" technique to enable their users to define data flow in compositions [9,23] and have shown its suitability in the context of end-user development [7]. However, the direct adoption of the technique to choreographed UI mashups is impossible due to the self-organizing nature of the aggregated widgets. The awareness and control facilities in this paper are inspired by the "wiring" approach and apply it to enable visualization and tailoring of the unified communication model.

6 Conclusions and Outlook

Missing understanding of inter-widget dependencies and lack of IWC control facilities can significantly impact usability and user experience within choreographed UI mashups. This paper presented an approach to systematically develop hybrid mashups with integrated IWC awareness and control mechanisms. The resulting solutions differ from the current state of the art in that they both enable self-emerging ("automatic" from end-users' point of view) IWC and keep users in control of how widgets communicate at the same time. The proposed reference architecture can be used as a guidance to build UI mashup platforms either from scratch or by extension of existing choreographed ones. One implementation of the reference architecture has been demonstrated in the context of an existing UI mashup platform based on publish-subscribe IWC strategy. The implementation is easily portable to other communication models such as RPC or shared memory.

A user study with 27 participants confirmed usability of the proposed control mechanisms and helped to discover shortcomings in the awareness ones.

⁹ <http://education.lego.com/en-us/preschool-and-school/secondary/mindstorms-education-ev3>

¹⁰ <http://www.ni.com/labview>

The drawn consequence for the future work is therefore to explore alternative non-ambiguous visualization techniques (e.g., flashing arrows instead of blinking icons) and to make the control mechanisms more prominent (e.g., by making the isolation icons accessible from the composition canvas). Finally, explanation of data being transferred between widgets has not been tackled sufficiently so far. The challenge here is to present the technical data (such as message syntax and semantic) in end-user-friendly way. Use of dedicated widget annotations or semantically enriched messages (as proposed in [18]) should be explored in the future.

Online Demonstration. A demonstration of the proposed awareness and control facilities integrated into Apache Rave and Apache Wookie projects is available at <http://vsr.cs.tu-chemnitz.de/demo/hybrid-ui-mashups>.

Acknowledgement. This work was supported by the European Commission (project OMELETTE, contract 257635). The authors thank Vadim Chepegin and TIE Kinetix b.v. for their contributions to the user study.

References

1. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE: A White Paper. EXP in Search of Innovation 3(3), 6–19 (2003)
2. Cappiello, C., Daniel, F., Matera, M., Picozzi, M., Weiss, M.: Enabling end user development through mashups: Requirements, abstractions and innovation toolkits. In: Piccinno, A. (ed.) IS-EUD 2011. LNCS, vol. 6654, pp. 9–24. Springer, Heidelberg (2011)
3. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A mashup environment for end user development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
4. Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Ignacio, J.: End-User-Oriented Telco Mashups: The OMELETTE Approach. In: WWW 2012 Companion Volume, pp. 235–238 (2012)
5. Chudnovskyy, O., Pietschmann, S., Niederhausen, M., Chepegin, V., Griffiths, D., Gaedke, M.: Awareness and control for inter-widget communication: Challenges and solutions. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 114–122. Springer, Heidelberg (2013)
6. Collis, J.C., Ndumu, D.T., Nwana, H.S., Lee, L.C.: The zeus agent building toolkit. BT Technology Journal 16(3), 60–68 (1998)
7. Imran, M., Soi, S., Kling, F., Daniel, F., Casati, F., Marchese, M.: On the systematic development of domain-specific mashup tools for end users. In: Brambilla, M., Tokuda, T., Tolsdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 291–298. Springer, Heidelberg (2012)
8. Isaksson, E., Palmer, M.: Usability and inter-widget communication in PLEs. In: Proceedings of the 3rd Workshop on Mashup Personal Learning Environments (2010)
9. JackBe. Presto Wires, <http://www.jackbe.com/products/wires.php>

10. Kirschenmann, U., Scheffel, M., Friedrich, M., Niemann, K., Wolpers, M.: Demands of modern pLEs and the ROLE approach. In: Wolpers, M., Kirschner, P.A., Scheffel, M., Lindstaedt, S., Dimitrova, V. (eds.) EC-TEL 2010. LNCS, vol. 6383, pp. 167–182. Springer, Heidelberg (2010)
11. Ko, A.J., Myers, B.A.: Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI Conf. on Human Factors in Computing Systems, vol. 6, pp. 151–158 (2004)
12. Krug, M., Wiedemann, F., Gaedke, M.: Enhancing media enrichment by semantic extraction. In: Proceedings of the 23rd International Conference on World Wide Web Companion, WWW 2014 Companion (to appear, 2014)
13. Lizcano, D., Soriano, J., Reyes, M., Hierro, J.J.: Ezweb/fast: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming ubiquitous soa. In: Proceedings of the 2nd Intl. Conf. on Mobile Ubiquitous Computing Systems, Services and Technologies, pp. 488–495. IEEE (September 2008)
14. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. *Simulation* 81(7), 517–527 (2005)
15. Minar, N., Burkhart, R., Langton, C.: The swarm simulation system: A toolkit for building multi-agent simulations. Technical report (1996)
16. Ndumu, D.T., Nwana, H.S., Lee, L.C., Collis, J.C.: Visualising and debugging distributed multi-agent systems. In: Proceedings of the Third Annual Conference on Autonomous Agents, AGENTS 1999, pp. 326–333. ACM, New York (1999)
17. North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: The Repast Symphony runtime system. In: Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago (2005)
18. Radeck, C., Blichmann, G., Meißner, K.: CapView – functionality-aware visual mashup development for non-programmers. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 140–155. Springer, Heidelberg (2013)
19. Rümpel, A., Radeck, C., Blichmann, G., Lorz, A., Meißner, K.: Towards do-it-yourself development of composite web applications. In: Proceedings of International Conference on Internet Technologies & Society 2011, pp. 330–332 (2011)
20. Viguera, G., Botia, J.A.: Tracking causality by visualization of multi-agent interactions using causality graphs. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 190–204. Springer, Heidelberg (2008)
21. Wilson, S.: Design challenges for user-interface mashups user control and usability in inter-widget communications (2012)
22. Wilson, S., Daniel, F., Jugel, U., Soi, S.: Orchestrated user interface mashups using W3C widgets. In: Harth, A., Koch, N. (eds.) ICWE 2011. LNCS, vol. 7059, pp. 49–61. Springer, Heidelberg (2012)
23. Yahoo! Yahoo! Pipes, <http://pipes.yahoo.com/>
24. Zuzak, I., Ivankovic, M., Budiselic, I.: A classification framework for web browser cross-context communication. CoRR, abs/1108.4770 (2011)