Mashups by Orchestration and Widget-based Personal Environments: Key Challenges, Solution Strategies, and an Application

Journal:	<i>Program</i>
Manuscript ID:	PROG-Feb-2012-0004.R1
Manuscript Type:	Article
Keywords:	Mashups, Widget Orchestration, Behavioral Patterns, Personal Environments, The Semantic Web, Workflow Mining

Mashups by Orchestration and Widget-based
Personal Environments: Key Challenges, Solution
Strategies, and an Application¹

Ahmet Soylu

KU Leuven, Kortrijk, Belgium

Felix Mödritscher

Vienna University of Economics and Business, Vienna, Austria

Fridolin Wild

The Open University, Milton Keynes, United Kingdom

Patrick De Causmaecker, Piet Desmet

KU Leuven, Kortrijk, Belgium

Abstract

Purpose – Mashups have been studied extensively in the literature; nevertheless, the large body of work in this area focuses on service/data level integration and leaves UI level integration, hence UI mashups, almost unexplored. The latter generates digital environments in which participating sources exist as individual entities; member applications and data sources share the same graphical space particularly in the form of widgets. However, the true integration can only be realized through enabling widgets to be responsive to the events happening in each other. We call such an integration widget orchestration and the resulting application mashup by orchestration. This article aims to explore and address challenges regarding the realization of widget-based UI mashups and UI level integration, prominently in terms of widget orchestration, and to assess their suitability for building web-based personal environments.

Design/methodology/approach – We provide a holistic view on mashups and a theoretical grounding for widget-based personal environments. We identify the following challenges: widget interoperability, end-user data mobility as a basis for manual widget orchestration, user behavior mining - for extracting behavioral patterns - as a basis for automated widget orchestration, and infrastructure. We introduce functional widget interfaces for application interoperability, exploit semantic web technologies for data interoperability, and realize end-user data mobility on top of this interoperability framework. We employ semantically enhanced workflow/process mining techniques, along with Petri nets as a formal ground, for user behavior mining. We outline a reference platform and architecture, compliant with our strategies, and extend W3C widget specification respectively - prominently with a communication channel - to foster standardization. We evaluate our solution approaches regarding interoperability and infrastructure through a qualitative comparison with respect to existing literature, and we provide a computational evaluation of our behavior mining approach. We realize a prototype for a widget-based personal learning environment for foreign language learning to demonstrate the feasibility of our solution strategies. The prototype is also used as a basis for the end-user assessment of widget-based personal environments and widget orchestration.

Findings – Evaluation results suggest that our interoperability framework, platform, and architecture have certain advantages over the existing approaches and proposed behavior mining techniques are adequate for the extraction of behavioral patterns. User assessments show that widget-based UI mashups with orchestration (i.e., mashups by orchestration) are promising for the creation of personal environments as well as for an enhanced user experience.

Originality/value – This article provides an extensive exploration of mashups by orchestration and their role in the creation of personal environments. Key challenges are described, along with novel solution strategies to meet them.

Keywords: Mashups, Widget Orchestration, Behavioral Patterns, Personal Environments, Foreign Language Learning, Embedded Semantics, RDFa, Ontologies, Petri nets, Workflow Mining

Paper type: Research paper

1 Page

¹ This article is a revised version of the following publication: Soylu, A., Wild F., Mödritscher, F., Desmet, P., Verlinde, S., De Causmaecker, P. (2011), "Mashups and Widget Orchestration", in MEDES'11, San Francisco, USA, ACM, pp. 226-234.

1. Introduction

The plethora of applications and information available on the Web is overwhelming and calls for efficient approaches to access and organize them. There has been a long-standing research on adaptive systems that can reflect on the needs and contexts of users. Acquired techniques have been applied successfully on the development of a variety of adaptive web systems (Brusilovsky and Maybury, 2002) providing context-tailored user experiences in terms of content (e.g., (Micarelli and Sciarrone, 2004)), as well as, presentation (e.g., interface (Hervas and Bravo, 2011)), and behavior (e.g., (Dang et al., 2008)). Adaptation and personalization, however, is only one side of the coin in which the user environment is considered to be a mere input for the user experience. On the other side, with a constructivist approach (Fosnot, 1996; Yang, 2001), Wild et al. (2008) consider the environment as an output of the user experience. Moreover, taking the open nature of context into account (Greenberg, 2001), it is impossible to define adaptation rules for all eventualities (Soylu et al., 2011a). In keeping with this perspective, providing end-users with appropriate means to reflect on their own state of affairs becomes an integral requirement. In this article, we are interested in the realization of personal (digital) environments. We define a personal environment as an individual's space of applications, data sources etc. in which she is engaged on a regular basis for personal and/or professional purposes. The individual at the centre modifies her environment through interacting with it, intending to positively influence her social, self, methodological, and professional competences and to change her potentials for future action and experience (Wild et al., 2008).

Considering traditional web experience, users are either tied to a single application (cf. (Knutov et al., 2009)) or they have to manage a set of applications, data sources etc. on their own with limited scaffolding support (e.g., iGoogle) or lack thereof (e.g., with bookmarking (van Harmelen, 2008)). To fill this gap, we consider exploiting the mashup approach. Although the idea of mashups is not new, nowadays it attracts researchers and practitioners more. This is mainly due to the shift and advancements in web technologies, such as Web 2.0, RESTful services, the Semantic Web, widgets etc. (cf. (Ankolekar et al., 2008; Baresi and Guinea, 2010; Sheth et al., 2007)). The mashup era has emerged in response to the challenge of integrating existing services, data sources, and tools to generate new applications and gained an increasing emphasis due to the ever-growing heterogeneous application market. In this respect, user interface (UI) mashups play a scaffolding role for building personal environments and can support cognitive processes, like fostering reflection, awareness, control and command of the environment. Nevertheless, the large body of work in this area focuses on service/data level integration and leaves UI level integration, hence UI mashups, almost unexplored. Mashups are usually realized either through a seamless integration, in which only the resulting application is known by the end-users, or through integration of original services, data sources, and tools, particularly in terms of widgets, into the same graphical space, in which participating applications and data sources are identifiable by the end-users. The former composes a unified functionality or data presentation/source from the original sources. The latter generates digital environments in which participating sources exist as individual entities. However, the true integration can only be realized through enabling widgets to be responsive to the events happening in each other. We call such an integration widget orchestration and the resulting application mashup by orchestration.

This article aims at exploring and addressing challenges regarding the realization of widget-based UI mashups and UI level integration, prominently in terms of widget orchestration (i.e., mashups by orchestration), and to assess their suitability for the creation of web-based personal environments. To this end, we provide a holistic view on mashups and a theoretical grounding for widget-based personal environments. We identify the following challenges: widget interoperability, end-user data mobility as a basis for manual widget orchestration, user behavior mining - for extracting behavioral patterns - as a basis for automated widget orchestration, and infrastructure. We introduce functional widget

interfaces (FWI) for application interoperability, exploit semantic web technologies for data interoperability, and realize end-user data mobility on top of this interoperability framework. We employ semantically enhanced workflow/process mining techniques, along with Petri nets as a formal ground, for user behavior mining. We outline a reference platform and architecture, compliant with our strategies, and extend W3C's widget specification respectively - prominently with a communication channel - to fill the standardization gap and to foster the re-usability of widgets and the development of standardized widget-based environments. We evaluate our solution approaches regarding the interoperability and infrastructure, through a qualitative comparison with respect to existing literature, and provide a computational evaluation of our behavior mining approach. We realize a prototype for a widget-based personal learning environment (WIPLE) for foreign language learning to demonstrate the feasibility of our solution strategies. The prototype is also used as a basis for the user assessment of widget-based personal environments and widget orchestration.

The rest of the article is structured as follows. Section 2 elaborates on the mashup landscape, widgets, widget-based personal environments, widget orchestration, and notable challenges. In Section 3, the related work is presented. The proposed solution strategies are described in Section 4 and evaluated and discussed with respect to relevant literature in Section 5. Finally, Section 6 concludes the article and refers to future work.

2. Mashups, Widgets and Personal Environments

The essence of UI mashups, for personal environments, is that they provide a unified interaction experience over a common graphical space for a collection of distributed digital entities and necessary affordances to blend functionalities of these entities and to stimulate exploration of new possibilities. Therefore, UI mashups are anticipated to play a key role for the realization of personal environments. Being completed with orchestration, they are intended not only to enhance but also to augment the end-user experience. There are particular characteristics which we deem important for personal environments based on UI mashups (Giovannella et al., 2010; Mödritscher et al., 2011; Soylu et al., 2010b): (1) open: a user can add/remove an entity to his environment at anytime, (2) clustered: a user can organize entities in his environment into different groups, (3) demand-driven: the behavior of the environment and entities are based on the explicit or implicit demands of the user, and (4) loosely coupled: entities available in the environment are independent from each other. In this context, we consider widgets, grounded on standards, as the building blocks of UI mashups and personal environments due to their promising suitability to meet the aforementioned characteristics. In what follows, a meta-level understanding of the overall picture, with a futuristic perspective, is depicted.

2.1. The mashup landscape

Mashups can be categorized into different types. We set forth two particular interlinked perspectives here. On the one hand, we categorize mashups into two types from an end-user point of view: the box type (cf. service/data mashups) and the dashboard type mashups (cf. UI mashups) (Soylu et al., 2010b). The former is realized through a seamless integration combining different applications and data sources into a single user experience, in which only the resulting application is known and perceived by the end-users. The latter is realized through the integration of original applications and data sources, particularly in terms of widgets, into the same graphical space (e.g., browser), in which participating applications and data sources can be perceived and identified by the end-users. On the other hand, from a technical point of view, we categorize mashups with respect to the source and integration approach as depicted in Figure 1. The source-wise categorization includes (1) service mashups (e.g., (Benslimane et al., 2008)), (2) data mashups (e.g., (Tummarello et al., 2010)), (3) tool mashups (e.g., (Kopecky et al., 2008)), and (4) hybrid mashups (e.g., (Soylu et al.,

2010a)). Service and data mashups are based on integration of services and data sources respectively. Tool mashups are similar to the service mashups; however, they are based on end-user applications with GUIs and the integration is carried out by extracting and driving the functionality of applications from their end-user interfaces (e.g., HTML forms). Hybrid mashups combine these three sources.

Figure 1. The mashup landscape: (1) service mashups by composition, (2) data mashups by composition, (3) tool mashups by composition, (4) hybrid mashups by composition of services, data, and tools, (5) mashups by orchestration, (6) applications with GUIs, and (7) widget run-time environment (client-side of a widget platform supporting execution of widgets - e.g., browser with a widget engine and API based on JavaScript).

4 Page

The integration-wise categorization is linked with and similar to the end-user perspective and includes (1) mashups by composition and (2) mashups by orchestration (see Figure 2).

Figure 2. Integration-wise categorization of mashups.

The difference with the end-user perspective lays in its emphasis upon the form of functional integration rather than the end-user perception. The former (i.e., by composition) composes a unified functionality or data presentation/source through a seamless functional integration of aggregated sources. The resulting mashup is a new application and participating sources are components of this new application. The latter (i.e., by orchestration) refers to the integration of aggregated resources into the same graphical space in a way that they are independent from each other in terms of functionality and interface. The source applications and data sources are aggregated possibly in terms of

widgets through a manual or automated widgetization process. In this respect, if the end application is a widget, an instance of mashup by composition might become an element of an instance of mashup by orchestration. The resulting mashup is a digital environment in which participating sources exist as individual entities. The true functional integration can only be realized through enabling widgets to be responsive to the events triggered by other widgets. We call such an integration ‘widget orchestration’. Our focus is on (semi-)automatic orchestration, that is enabling widget platform to learn user behavioral patterns through harnessing event logs and, upon initiation of a pattern, (semi-)automatically executing the corresponding pattern flow, i.e., an ordered, parallel etc. set of widget actions. The automation process also requires the communication of relevant data through the flow.

2.2. Widget-based UI mashups for personal environments

The idea of widgets has existed in various forms such as badgets, gadgets, flakes, portlets etc., and differentiates with respect to the underlying technology, the availability of backend services and so on. In this paper, we are interested in web widgets. Typically, a web widget (cf. (Cáceres, 2011; Zhiqing et al., 2010)) is a portable, self-contained, full-fledged, and mostly client-side application. It is hosted online and provides a minimal set of functionality (with/without backend services) through considerably less complex and less comprehensive user interfaces. Widgets are expected to be re-usable, which is achieved by enabling widgets to be embedded in different platforms satisfying certain standards and specifications (e.g., W3C widget specifications (Cáceres, 2011)). Various technologies can be used to implement widgets (notably HTML and JavaScript, Java Applets, Flash etc.), but cross-platform and device support is crucial due to re-usability considerations.

Figure 3. Widgetization of an example application.

Some widgets are developed for generic purposes such as clock, calendar etc. widgets for specific platforms (e.g., for Windows 7), mostly, without re-usability and portability concerns. More advanced widgets are developed for specific purposes either from scratch or as a micro version of already existing applications (see Figure 3). As an example of the latter, Figure 3 shows a web application called ‘mediatic’ (<http://www.kuleuven-kulak.be/mediatic/> - a foreign language learning tool providing video materials) and its widgetized form that we have developed. A widget platform is required to support execution of multiple widgets over a common space. A widget-based environment can be populated by an end-user or pre-

populated by a programmer or super-user. Widgets can be populated into a single space, or multiple working spaces can be created to cluster related widgets (e.g., one space for language learning, one for entertainment etc.).

During a normal web navigation experience a typical user finds, re-uses, and mixes data by switching between different web applications being accessed either from different browser instances or tabs. Normally, every regular web user generates her own portfolio of applications (implicitly or explicitly - e.g., through bookmarking (van Harmelen, 2008)) over time. Therefore, for a regular user, one can expect to observe behavioral patterns representing regularly performed actions and activities over the applications in her portfolio (e.g., a user watches videos in a video sharing application and publishes the ones she likes in a social networking application). Widget-based environments can facilitate such a user experience by allowing users to access their portfolios through a common graphical space, where each application or data source is represented as a widget, and allows users to create their own personal environments. At a conceptual level, a web-based personal (digital) environment (cf. (Friedrich et al., 2011; Jongtaek and Haas, 2010; Severance et al., 2008)) can be seen as a user-derived sub-ecosystem where member entities come from the Web (the ultimate medium for the supreme digital ecosystem). This is because, indeed, personal environments can be intertwined with the physical world, since the member entities are not limited to web applications and digital data sources anymore. It includes any physical entity, e.g., devices, people etc., having a digital presence (cf. (Kindberg et al., 2002)). A variety of devices, like mobile phones, tablet PCs, intelligent household appliances, etc. are expected to be connected to the Internet (or to local networks through wired/wireless technologies like Bluetooth etc.) and serve their functionalities through Web and Web-based technologies, e.g., via RESTful APIs (Dillon et al., 2011; Dillon et al., 2009). Therefore, various Internet-connected devices can be part of the personal environments through widgets acting as a medium of virtual encapsulation. This allows users to merge their physical and digital environments into a complete ecosystem (i.e., personal and pervasive) and to organize interaction and data flow between them (Soylu et al., 2010a).

6 Page

In widget-based personal environments, the user experience can be enhanced in several ways, notably, (1) by enabling data, provided by a user or appearing as a result of her actions in a widget, to be consumable by other widgets, particularly, in terms of end-user data mobility (i.e., enabling an end-user to copy content from one widget to another effortlessly) and (2) by automating the execution of regular user actions by learning user behavioral patterns from the logs generated as a result of users actions. We see such interplay between widgets - or web sources in general - as an orchestration process. Widget orchestration can happen in various forms in a widget-based environment. We consider followings of crucial importance: (1) user-driven: users manually copy data from one widget to the other and initiate the target widgets. The manual process can be enhanced through facilitating the end-user data mobility (e.g., select, copy, and paste with drag & drop from widget to widget), (2) system-driven: the system learns user behavioral patterns by monitoring events (each corresponds to a user action) and data emerging as a result and then handles data mapping, transportation, and widget initiation processes (semi-)automatically, (3) design-driven: a programmer, a super-user, or even an end-user pre-codes widget behaviors, e.g., which widget should react to which event and how, and (4) hybrid: similar to the system-driven orchestration, the system learns user behavioral patterns; however, the control is shared (cf. (Spiekermann, 2008; Valjataga and Laanpere, 2010)) by means of user mediation, recommendations etc. For instance, instead of going for an immediate automation, the user is provided with recommendations and the automation only happens if the user decides to follow a particular suggestion. This paper focuses on system-driven widget orchestration.

However, system-driven widget orchestration faces us with several challenges. (1) Widget interoperability: (a) application interoperability - in order to enable widgets to be responsive to the user actions happening in other widgets, a loose functional integration is necessary. Since widgets are developed by different independent parties, standards and

generic approaches are required to ensure loose coupling, (b) data interoperability - widgets need to share data, particularly during a functional interplay. Since widgets do not have any pre-knowledge about the structure and semantics of data provided by other widgets, standards and generic approaches are required to enable widgets to consume data coming from other widgets. (2) User behavior mining: each user action within widgets can be represented with an event. Algorithms that are able to learn user behavioral patterns, i.e., structure (topology) and routing criteria, from the logged events and circulated data are required along with a formal representation paradigm to share, analyze, and present behavioral patterns. (3) Infrastructure: the abovementioned challenges require any platform, on which the widgets run, to be part of possible solution strategies (e.g., how heterogeneous applications communicate events and data). Standardization will enable different communities and parties to develop their own compliant widgets and platforms. It will enable end-users to create their own personal environments by populating heterogeneous applications and data sources and by orchestrating them. In this respect, specification of a generic communication channel for widgets is of crucial importance to enable integration. The aforementioned challenges are also in the core of the successful realization of other widget orchestration strategies and widget-based environments in general, notably, user-driven orchestration through end-user data mobility.

3. Related Work and Discussion

There exists a considerable amount of work on mashups by composition due to the popularity of web service composition, e.g., (Benslimane et al., 2008; Kopecky et al., 2008; Sheth et al., 2007; Taivalsaari, 2009; Tummarello et al., 2010); however, there is a limited work on UI mashups. At this point, we remark that approaches using visual constructs (e.g., widget like) and programming support for the construction of mashups by composition (e.g., service, data etc.) should be distinguished from UI mashups (e.g., (Baresi and Guinea, 2010; Cappiello et al., 2011)). In conjunction with the rising popularity of widgets and W3C widget specifications (e.g., (Cáceres, 2011)), the use of widgets for UI mashups and personal environments gained attraction (e.g., (Govaerts et al., 2011; Nelkner, 2009; Sire et al., 2009; Soyly et al., 2011b)). Yet, first examples are proprietary such as Yahoo widgets, Google gadgets, Open Social gadgets etc. and integration is mostly limited (i.e., functional interplay). In more advanced approaches, particularly, Intel Mash Maker (Ennals et al., 2007), mashArt (Daniel et al., 2009), and Mashlight (Baresi and Guinea, 2010), IU mashups are developed manually by a developer or a skilled user, notably with visual programming support (cf. (Shu, 1989)). The main problems with these approaches are that they are mainly design-driven (including data mappings, flow structure and routing) and not truly appropriate for naive end-users. Another handicap is that they mostly provide limited compliance on the W3C's widget specification family or do not describe any possible extension.

Regarding approaches based on widget interoperability, in the current literature, the trend is towards inter-widget communication (IWC) (Friedrich et al., 2011; Govaerts et al., 2011; Laga et al., 2010; Nelkner, 2009; Pohja, 2010; Sire et al., 2009; Wu and Krishnaswamy, 2010) in which basically an event is delivered to relevant widgets (i.e., unicast - to a single receiver, multicast - to multiple receivers, or broadcast - to all possible receivers) whenever a user performs an action inside a widget. Multicast and broadcast are the basis of the IWC. In the former, widgets subscribe to particular events and/or particular widgets etc. and get notified accordingly while, in the latter, events are delivered to all widgets in a platform. In both cases, the receiving widgets decide whether and how they react to an event depending on the event type, content, etc. However, there exist some problems in the IWC. First of all, since widgets have to decide on which events to react and how, they are overloaded with extra business logic to realize responsiveness. Secondly, responsiveness is hard to be realized. Either widgets have pre-knowledge of each other, and hence semantics of the events they deliver, or widgets exhibit responsiveness through matching syntactic signatures

of the events delivered. The former approach is not realistic because widgets are developed by different parties and in a broad and public manner. The latter is problematic in terms of its success, since syntactic signatures are simply not enough for a successful identification of relevant events. Thirdly, since each widget acts independently, without any centralized control, it is unlikely to achieve a healthy orchestration. Chaotic situations are most probably to arise in an open environment when several self-determining widgets respond to the same/different events in a distributed and uncontrolled way.

Wilson et al. (2011), in their recent work that partially coincides with our predecessor study (cf. (Soylu et al., 2011b)), define three UI mashup models supporting widget responsiveness, namely Orchestrated UI mashups, Choreographed UI mashups, and Hybrid UI mashups. Orchestrated UI mashups refer to the case where interactions between widgets are defined explicitly and managed centrally. Event (notification of a user action in a widget) – operation (a widget behavior triggered by an event originating from another widget) mappings as well as data transformations and mappings are pre-defined by a developer (please note the conceptual difference that we have, that is by mashups by orchestration we refer to the general idea of interplaying widgets regardless of how it is realized). Choreographed UI mashups refer to the case where interactions between widgets (i.e., responsiveness) emerge from the individual decisions of widgets. This is achieved through ensuring that each widget complies with a reference topic ontology. Each widget publishes its events with respect to a reference topic ontology and widgets subscribe to the events of interest. Hybrid UI mashups refer to the case where widgets maintain a partial autonomy, that is, the interplay (i.e., responsiveness) between widgets is constrained through the central logic programmed by a developer. These three models inherit previously elaborated drawbacks for possible applications of the IWC approach. Regarding the first model, a central logic, pre-coded by a developer, constraints the open and demand-driven characteristics of personal environments in terms of user experience, that is although new widgets can be added to the environment by an end-user, it cannot truly be part of the user experience before a developer describes its role in the orchestration process. Moreover, the approach becomes inflexible since a developer is required to describe event – operation mappings as well as data mappings and transformations. The second model, being based on a reference topic ontology, is better than a purely syntactic approach, though it does not comply with the demand-driven characteristic of personal environments. One should also take it into consideration that a semantic match between an event and an operation does not guarantee that the emerging interplay is sound. Lastly, a distributed orchestration approach (with or without semantics) complicates the widget development as previously mentioned. Entry barriers for the widget development should be kept minimal. The third model maintains the drawbacks described for the previous two models and is inflexible since it requires a developer to be under full possession of the environment and the widgets.

In the current examples of widget-based environments, e.g., (Friedrich et al., 2011; Govaerts et al., 2011; Nelkner, 2009; Wild et al., 2008), the idea of interplay between widgets already exists; however, it is either pre-designed or purely based on syntactic or semantic similarities between widgets. Behavioral patterns, which are necessary to comply with the demand-driven nature of personal environments in an automated approach, are not exploited and, due to that, a formal ground for mashups by orchestration is not explored yet. In Srbljic et al. (2009), a widget-based environment (i.e., mashups by orchestration) is used for the end-user programming of composite applications (i.e., mashups by composition). The work is particularly relevant since it aims at empowering end-users to program by demonstration which requires learning from end-user behaviors. Each source is represented as a widget and an end-user performs a set of actions over these widgets to achieve the outcome she desires. The actions of the user are monitored and a composite application is generated respectively. The algorithm employed corresponds to a part of a well known workflow mining algorithm (α -algorithm), yet a formal modeling instrument, such as Petri nets, is not utilized. Data mappings as well as the topology of composite

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

application flow (e.g., parallel, sequence etc.) are provided manually by the end-user (with visual programming support tailored for skilled users). Coming back to Mashlight (Baresi and Guinea, 2010), though being a design-driven approach, in terms of the grounding formalism, the authors employ a process model based on a directed graph. Indeed, their work is rather an example of mashups by composition based on widgets displayed in a sequential order. The authors later introduce super-widgets which are indeed containers for multiple widgets activated in parallel. The problem with this approach, in our context, is that their grounding model is proprietary and requires means for validation, verification, and sharing of patterns. In a personal environment based on widgets, facilities related to the data mobility should be designed for naive end-users while topology and routing criteria should be extracted implicitly. A formal ground is a must for validation, verification and sharing of the behavioral patterns to avoid emergence of pathological patterns, to enable possible share between users, and for visualization of the extracted patterns.

Ngu et al. (2010) propose an approach that allows composition of Web-service-based and non-Web-service-based components, such as web services, web applications, widgets, portlets, Java Beans etc. Further, they propose a WSDL based approach, enriched with ontological annotations, to describe programmatic inputs and outputs of components in order to allow searching and finding complementary and compatible components. The overall approach allows users to progressively find and add components to realize a composite application through wiring the outputs and inputs of different components. The approach is based on IBM Lotus Expeditor which includes a Composite Application Editor (CAE) and Composite Application Integrator (CAI). CAI is the run-time editor which reads and manages the layout information of composite application and responsible of passing messages between components. The CAE allows assembling and wiring components. The approach presents components with UIs in a common graphical space and follows a data-flow-oriented approach rather than a task-oriented approach (i.e., mashup by composition - which aims at putting together a single business process for the purpose of automating a specific task). In other words, there is no specific begin and end-state, a component can start executing whenever it receives the required input, and there is no explicit control flow specified. Although the matching mechanism and the way components are put together are still more task-oriented, the data-flow oriented perspective matches the characteristics of personal environments. However, firstly, the presented approach relies on the users for designing mashups and interaction of components. Secondly, the approach do not present any specification regarding event delivery and communication; hence it remains ad-hoc. The proprietary nature of the editing and run-time environment hinders the possibility of wide acceptance of the resulting composition framework and remains weak against more ubiquitous, simple, and standard approaches, e.g., mashups, based on W3C widgets, which can simply run on any standard browser. Thirdly, the proposed approach does not provide any formal means for validation and verification of the compositions to prevent the emergence of pathological mashups. Finally, in the current shape, the proposed approach is more appropriate for creation of task-oriented mashups, mixing functional and interface integration, for enterprises and skilled users rather than personal environments for the end-users.

Regarding the architecture, the existing work is mainly repository-centric (Friedrich et al., 2011; Nelkner, 2009). The Apache Wookie server (<http://getwookie.org/>) is notable in this respect. Wookie does not only host W3C widgets, but also provides basic services such as inter-widget communication (over a server-sided communication mechanism), preference management etc. Widgets access services, which are provided by the widget server, through containers in which the server places them before the delivery. Such a centralized approach is inflexible and overloads the repository by aggregating services and tasks, which should normally be provided by a client-side run-time system, to itself. Such an approach is not appropriate for a heterogeneous environment since widgets coming from different repositories cannot communicate. We believe that any possible architectural decision should

be taken in compliance with existing specifications. Prominently, the W3C's widget family of specifications provides a set of standards to author, package, digitally sign, and to internationalize a widget package for the distribution and the deployment on the Web. Yet, the W3C's widget specification and the standardization process still being active, there remains major room for extensions. More specifically, with respect to the challenges described in Section 2.2, extensions are required for communication infrastructure, event delivery, functional integration, and data mobility. In Wilson et al. (2011), the authors propose to extend the W3C's widget specification family with communication support and also with means to enable widgets to disclose their functionality for programmed orchestration support. We agree upon these extensions as asserted by our earlier work (Soylu et al., 2011b) with the difference that we extract control logic implicitly from the user interactions. These extensions will be described in detail in Section 4.

4. Solution Strategies

Traditional UI mashups (e.g., (Baresi and Guinea, 2010; Ngu et al., 2010)) are usually compositional and enterprise-oriented. They compose Web APIs and data feeds into new applications and data sources to typically serve specific situational (short-lived) needs of the users in a task-oriented manner (cf. (Yu et al., 2008)). Each mashups by composition always have a particular task as a goal and require some manual/automated development process. Considerable amount of effort has been spent towards creating development environments to enable effortless composition, with support for data mapping, flow operations, user interface combination etc., addressing skilled users to novice users (cf. (Di Lorenzo et al., 2009)). However, for personal environments, mashups should be employed for the orchestration of a dynamic and heterogeneous set of applications, with respect to the active data-flow, in a user-oriented manner. This is comparable to traditional desktops where users run a set of applications and manually blend their functionalities. Therefore, mashups by orchestration follow a experience-oriented perspective and allow users to populate various applications and to orchestrate them spontaneously. There is not only a single task in mind. There is no specific start operation, ending operation, pre-defined control flow etc. In this respect, our task is not to come up with a development environment; however, what we target is a platform (e.g., operating system) which allows aggregation of applications in form of widgets and enables user-oriented widget interaction. We first provide an interoperability framework; the interoperability framework is combination of specifications for functional interfaces for widgets and semantic annotation of content, events, and interfaces in order to address application and data interoperability considerations respectively. Secondly, we propose an end-user data mobility facility, built on top of the interoperability framework, to enable the user-driven data exchange between widgets. This is particularly important since, contrary to data exchange between services, output of an application with UI (a widget in our case) is not necessarily programmatic and not all the returned content is relevant to the need of end-user. Thirdly, we propose and specify a communication channel and standard run-time services (e.g., event delivery, preference management etc.) for a widget platform, along a reference architecture to facilitate rapid realization of widget-based personal environments. Finally, we provide an algorithmic solution to enable widget platform to learn commonly executed end-user patterns to automate the interplay between widgets with respect to the events occurring as a result of user interactions at UI level.

Our perspective, for personal environments, is to enable end-users to populate their own spaces and to organize and orchestrate the available entities with respect to their own changing needs. Expectedly, a design-driven model and others have their places as well (e.g., when specific experiences are required to be designed). However, with a fundamentalist approach, we first define the most generic and suitable model on which more specific models can be built. Otherwise, results are more likely to be proprietary. In this respect, the realization of an orchestration model with an open and demand-driven characteristic allows

more specific models and design tools (e.g., design-driven) to be realized upon by imposing new constraints (cf. (Baresi and Guinea, 2010)). Here, the path we follow is to first empower end-users with generic facilities to realize user-driven orchestration, later to enable system to extract behavioral patterns from the user interactions to realize system-driven orchestration, and finally to enable the system to mediate with the end-user and/or to provide recommendations in order to realize a facilitated orchestration experience (cf. (Spiekermann, 2008) - note that the latter is not within the scope of this article).

Technically, to realize our approach, we first empower the end-users with facilities to communicate data from one widget to another (i.e., data mobility). Each widget notifies the platform, through a communication channel, whenever a user action occurs, including data exchanges. The platform stores events into the event log and monitors the log for a certain time to extract behavioral patterns. A behavioral pattern is a partial workflow with a flow structure and routing criteria. We define functional interfaces, which allow widgets to disclose their functionalities, so that the platform can automatically execute extracted patterns. The use of domain knowledge, along ontological reasoning support, and standard vocabularies, for enhancing event signatures, functional interfaces, and widget content including interactional elements (e.g., forms), improves the pattern mining (i.e., extraction) process as well as data mobility. We prefer to use and extend the W3C's widget family of specifications due to ubiquity, simplicity, and device-natural characteristics of its underlying technology (i.e., HTML, JavaScript), yet the overall strategy and proposed approaches remain generic. The foremost advantage of our approach is that widgets and widget development remain simple and complicated orchestration tasks are delegated to the platform. One can also build a design-driven model or a widget-driven distributed orchestration model on top of the main instruments of the proposed model (i.e., functional interfaces, data mobility and communication infrastructure) while keeping in line with a standard-oriented approach (cf. (Baresi and Guinea, 2010; Govaerts et al., 2011; Wilson et al., 2011)).

4.1. Widget interoperability and end-user data mobility

Regarding the application interoperability, the proposed strategy is that widgets disclose their functionalities through standardized client-sided public interfaces (e.g., JavaScript APIs) which we call functional widget interfaces (FWI) as shown in Figure 4.

Figure 4. Functional Widget Interfaces (FWI).

FWI allows the corresponding platform to control widgets through functional interfaces. Each function corresponds to a user action within a widget that generates an event when triggered. Event notifications and control requests are communicated between the platform and the widgets, through a communication channel, over services provided by the run-time system of the platform (see Section 4.2). Widgets can share the functionality of their APIs with the platform through a handshake process performed over a standard interface

function (e.g., with WSDL) or it can be extracted from the event logs. The latter requires functionality provided with GUIs and APIs to be identical. The former is required for a design-driven approach where functionality of the widgets should be available to the users (e.g., programmer) directly.

Figure 5. Widget triggered by an event of another widget.

An example is given in Figure 5. In this example, there are two widgets in a user's environment, namely, 'mediatic' and 'flickr' (a widget that we have developed for a Web 2.0 tool that is used to store, sort, search and share photos online – see www.flickr.com). The user watches a video material from the 'mediatic' widget with sub-titles, and when clicking on certain words of the text (e.g., the word 'car' in Figure 5), the 'mediatic' widget delivers an event to the platform. The platform decides on an appropriate widget to react on this event based on learned patterns. In this case the 'flickr' widget is selected. The relevant event data are extracted and communicated to the 'flickr' widget with the desired functionality. The 'flickr' widget executes the request by fetching and displaying images relevant to the word of interest.

12 Page

Figure 6. Use of domain knowledge for data interoperability - (1) HTML forms, (2) functional interfaces, (3) events, and (4) content (non-interactive).

Concerning the data interoperability, the use of domain knowledge or generic/domain-specific vocabularies enhances interoperability as well as end-user data mobility. For instance, in Figure 5, the ‘mediatic’ widget announces an event informing that the noun ‘car’ is clicked and the ‘flickr’ widget is selected to respond although it accepts strings, which are of the word type. This is because, an ontological reasoning process asserts that the noun ‘car’ is an instance of the class ‘word’ since the class ‘noun’ is a subclass of the class ‘word’ as declared in the grounding ontology. A semantic approach also enhances the behavior mining as described in Section 4.3. For this purposes, on the one hand, we enhance events and function signatures with domain ontologies or vocabularies as shown in Figure 6. On the other hand, we annotate widget content including interactional elements (e.g., forms) with domain knowledge in order to enable end-users to copy content from one widget to another by simple clicks (i.e., data mobility). Each annotated content piece is visually marked to support end-users with necessary affordance cues. An example is depicted in Figure 7, for two widgets, namely, ‘dafles’ (a widget that we have developed for an online French dictionary – see <http://ilt.kuleuven.be/blf/>) and ‘dpc’ (a widget that we have developed for an online multilingual parallel corpus – see <http://www.kuleuven-kulak.be/DPC/>). A user looks up for the meaning of a French word in the ‘dafles’ widget and decides to see example sentences as well as their English translations. Therefore, she clicks on the marker of one of the items (data chunk) of the result list, and copies that item to the ‘dpc’ widget by clicking on the marker of the target form.

Figure 7. Data is copied from one widget to another by an end-user. ‘Zoek’ stands for ‘Search’, ‘Taal’ stands for ‘Language’, and ‘Woordsoort’ stands for ‘Word type’.

We use embedded semantics technologies for in-content annotation (cf. (Adida, 2008; Bizer et al., 2009)), e.g., microformats, RDFa, microdata, eRDF etc. They can be used for structuring content (i.e., with types and data type properties), interlinking content elements (cf. (Bizer et al., 2009) - i.e., a form of linked-data - with object type properties), and embedding high-level domain semantics (e.g., class – subclass relationships). Figure 8 and Figure 9 show excerpts from annotated HTML content (with RDFa, cf. (Adida, 2008)) of the ‘dafles’ and ‘dpc’ widgets and extracted semantic data in simplified N-Triples format. The excerpt shown in Figure 8 and Figure 9 belongs to the user-selected data item (cf. the left-hand side of the Figure 7), and the target HTML form (cf. the right-hand side of the Figure 7) respectively. The visual marking of annotated content including interactional elements is

handled through a specific widget plugin that we have developed. The plugin observes content changes and, upon each change in content, marks annotated content pieces. Each marking is associated with a standard event described in what follows.

```

HTML: dafles
1 <span about="www.dafles.com#r1" typeof="ll:verb">
2   <span property="ll:text">abandonner</span>
3   <span property="ll:language">FR</span>
4 </span>

N-Triple: dafles
5 <dafles:r1> <rdf:type> <ll:verb>.
6 <dafles:r1> <ll:text> "abandonner".
7 <dafles:r1> <ll:language> "FR".

```

Figure 8. Semantic data is extracted from an annotated HTML content (source).

```

HTML: dpc
1 <span about="param:p" typeof="ll:word">
2   Zoek:
3   <div rel="ll:text" resource="param:woord">
4     <input id="woord" type="text"/>
5   </div>
6   Taal1
7   <div rel="ll:language" resource="param:taal1">
8     <select id="taal1">
9       <option value="EN">EN</option> ...
10    </select>
11  </div> ...
12  Woordsoort
13  <div rel="rdf:type" resource="param:woordsoort">
14    <select id="woordsoort">
15      <option value="noun">Noun</option> ...
16    </select>
17  </div>
18 </span>

N-Triple: dpc
19 <param:p> <rdf:type> <ll:word>.
20 <param:p> <rdf:type> <param:woordsoort>.
21 <param:p> <ll:text> <param:woord>.
22 <param:p> <ll:taal> <param:taal1>.

```

Figure 9. Semantic data is extracted from an annotated HTML form (target).

In order to copy a user-selected data chunk from a source widget to a target widget, a special event 'dataSelected' (i.e., copy) is introduced to inform the platform. This standard event is only associated with the markings of non-interactive content pieces and communicates the selected data chunk as an event payload. The extracted data indeed forms a small RDF graph. Later, the user clicks on the marker of the target HTML form, and the target widget informs the platform with a special event 'formSelected' (i.e., paste). This standard event is only associated with the markings of interactive content pieces and communicates the data extracted from the target content piece (i.e., HTML form) as an event payload. The extracted data indeed can be represented as a (partially) empty graph.

Consequently, data mobility is achieved through graph matching (see Figure 10). In order to do so, we first transform the empty graph into a SPARQL query (cf. Figure 10). We introduce a specific name space (<http://itec-research.be/ns/param> with prefix 'param') to define variable type resources in order to annotate form fields as shown in Figure 9. This is an open and empty name space. Identifiers of the variable type can be defined under this name space at the time of authoring and the scope of each identifier is limited to the subject document. It is required that the form author keeps variable resource identifiers and corresponding form field identifiers identical. In this respect, the transformation starts with converting each variable type resource (e.g., "<param:p>") in each RDF statement of the triple set into SPARQL variables (e.g., "<param:p> <rdf:type> <ll:word>" becomes "?p rdf:type ll:word."). This transformed triple set, as a whole, is used to construct the WHERE clause of the SPARQL query and, finally, we construct the SELECT clause from the variable type objects of each RDF triple. We execute the resulting SPARQL query over the first RDF graph with ontological reasoning support. As a result, the empty graph is matched with the former RDF graph and the values of the form fields are set with the data matched from the source widget. In Figure 8 and Figure 9, the 'll' prefix stands for the namespace associated with the sample language learning ontology.

Figure 10. The target HTML form is transformed into a SPARQL query for graph matching.

Once the run-time platform resolves the target form field values, it communicates these values to the target widget through a specified FWI function (see Section 4.2 for details). This widget specific function sets the values of form fields respectively by exploiting the fact that variable resource identifiers and form field identifiers are identical; however, one can also exchange HTML form identifiers within event and control messages for more complex widgets (e.g., when there is more than one HTML form in a single interface etc.). Multiple paste events can be executed over the extracted source data (i.e., can be copied into several target forms) as far as data is not overwritten by a new copy event.

4.2. Platform, framework and architecture

The platform is composed of two primary layers (see Figure 11), namely a run-time system and a backend system. The run-time system resides at the client (e.g., browser) and is responsible for the operational tasks and the delivery of standard platform services (e.g., preference management, cf. (Cáceres, 2011)) to the widget instances. The backend system resides at the server side and is responsible for the persistence and decision-making.

Figure 11. The platform architecture – (1) run-time system/environment, (2) backend system, and (3) the Web.

The run-time system and backend system are composed of different components. Regarding the run-time system: (1) Widget containers (e.g., a HTML frame), in our context, hold widget instances in the user space and bridge communication ends of widget instances and the environment. Triggers for basic facilities, related to the presence of a widget instance in the environment such as remove, close, minimize, pin, move etc., are attached to the containers. (2) The environment controller manages presence related facilities, such as (absolute/relative) widget positioning, for widget instances over the widget containers and is responsible for the introduction of new sub-spaces, repositories, and widget instances (widgets from repositories or standalone widgets from the Web). (3) The communication channel allows bidirectional communication between the widget instances and the

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

environment. Widget instances communicate events, preferably preferences and data access requests as well, to the platform, and the platform communicates data and control commands for orchestration to the widget instances through the communication channel. (4) The run-time system core provides standard system services to the widget instances, particularly, for preference management through (4a) the preference management service, for event delivery through (4b) the event management service, and for data access requests to widget backend services through (4c) the data access service using (5) the proxy agent. The core coordinates the orchestration through (4d) the adaptation controller by submitting control commands to the widgets over the communication channel. The adaptation controller handles data mediation and transportation and can utilize a light-weight (6) client-side reasoner for this purpose (e.g., JSW toolkit <http://code.google.com/p/owlreasoner/>, EYEClient <http://n3.restdesc.org/rules/generalized-rules/>). The adaptation controller can also submit re-positioning requests to the environment controller (e.g., in order to move involved widgets closer in course of an active interplay). We consider this facility particularly important, since there exists some interdependencies between location expectations of widgets as suggested by Gali and Indurkha (2010).

Regarding the backend system components, (1) the manager handles preference persistence through (1a) the preference manager and the state of the environment (e.g., widgets, widget positions etc.) through (1b) the widget manager. (1c) The context manager stores event logs and any other contextual information for context based adaptation (cf. (Bettini et al., 2010)). (1d) The adaptation manager decides on adaptation rules (i.e., control commands), particularly through learning behavioral patterns, and submits them to the adaptation controller. It utilizes a (2) server-side reasoner. (3) The proxy is responsible of retrieving data from external data sources (mainly from widget back-end services) upon receiving a dispatch request, initiated by a widget, from the proxy agent of the run-time system.

We prefer to detail the specification of important components of the platform and architecture by providing examples from our own prototype. The communication channel (see Figure 12) and standard services require special attention since they need to be standardized while other components are specific to a platform. The communication channel constitutes the backbone of the platform and the personal environment. For approaches allowing direct widget communication, the communication channel allows communication between local and remote widgets through the platform. Run-time system services (e.g., preference, data access, event delivery etc.) are mainly built on top of the communication channel, hereby allowing us to come up with a non-complex and generic platform and architecture. The communication channel consists of two ends that is a run-time end and a widget end (for each widget). In Figure 12, the widget end of the communication channel is shown. The run-time end of the communication channel is similar to the widget end. The communication channel is based on the 'window.postMessage' method of HTML 5 allowing cross-origin communication (realize that widget sources are mostly distributed). It provides a method named 'channel.send' for event and request delivery (e.g., preference, data access etc.). This method accepts two arguments, a 'receiver' and a 'message'. In our context, the receiver is only the parent (that is the platform) for a widget; however, for other models, one can also distribute events to a specific widget or to a set of subscribed/all widgets.

```

1  Communication Channel
2  1 var Channel = function(){
3      this.send = Send;
4      this.addCallListener = AddCallListener;
5  }
6
7  function Send(receiver, message){
8      if(receiver == 'parent' )
9          window.parent.postMessage(message, '*');
10     else if ... // to subscribed/all widgets
11     else if ... // to a specific widget
12 }
13
14 function AddCallListener(){
15     var onmessage = function(e){
16         var message = e.data;
17         var origin = e.origin;
18         . . .
19         var targetFunction = extractTarget(message);
20         . . .
21         var fn = widget.shortName+'.'+targetFunction;
22         eval (fn) (message);
23     };
24
25     if (typeof window.addEventListener != 'undefined'){
26         window.addEventListener('message', onmessage, false);
27     } else if (typeof window.attachEvent != 'undefined') {
28         window.attachEvent('onmessage', onmessage);
29     }
30 }
31
32 channel = new Channel();
33 channel.addCallListener();

```

Figure 12. Widget end of the communication channel.

The message argument is indeed composite and represented in JSON format as key-value pairs, see Figure 13. A message is composed of a message body and a payload. Regarding the message body, a message type is mandatory. In our context, the following message types are required to be used by a widget: 'event', 'preferenceSet', 'preferenceGet', 'handShake', and 'access'. Regarding the platform, 'control', 'preference', 'handshake', and 'dispatch' message types are required. A return function and/or target function might be required depending on the message type. The latter specifies a function/procedure (i.e., name or alias) in the receiver application that needs to take care of the received message while the former specifies a function in the sender application that the target application must return its response to. The message types are described as follows; unless otherwise noted target and return functions are not required. The 'event' type messages are used to deliver user actions in widgets and an event name needs to be specified by using 'messageName' key. There are two special events with reserved names 'dataSelected' (i.e., copy) and 'formSelected' (i.e., paste) as stated in Section 4.1. A return function needs to be specified, for only the event named 'formSelected', in order to deliver values for matched data items. The 'preferenceSet' and 'preferenceGet' are used for the preference persistence, that is, for storing and retrieving respectively. The 'handShake' message type is used by a widget to deliver its FWI function set to the platform. The 'access' message type is used by a widget for data access (through the proxy of platform); a return function needs to be specified for data dispatching. The 'control' type messages are used by

the platform to send control commands to widgets; a target function needs to be specified for control messages. The ‘preference’ type messages are used by the platform to deliver stored preference values to the requesting widgets. The ‘handshake’ message type is used by the platform to request the function set of a widget’s FWI. Finally, the ‘dispatch’ message type is used by the platform to return the requested data to the requesting widget; a target function is required.

```

1  Message format
2  { // message body
3    "message":{
4      "messageType":"type of message",
5      "messageName":"name of the message",
6      "returnFunction":"return function name",
7      "targetFunction":"target function name",
8      "data":[ // message payload
9        { "key":"value" },
10       . . .
11       {
12         "entity":[
13           {
14             "?x":{ URI },
15             "?y":{ URI },
16             "?z":{ URI OR
17               "label":"a label",
18               "lang":"lang",
19               "dtype":"dtype"
20             },
21           . . .
22         ]
23       },
24       . . .
25     ]
26   }
27 }
```

Figure 13. Message format for communication.

Regarding the message payload (i.e., data), it follows the same key-value approach; however, the message payload is not limited to a syntactic key-value structure, particularly for ‘event’ and ‘control’ type messages, as explained in Section 4.1. Semantic data (i.e., in form of typed entities) can be exchanged as well with the reserved key ‘entity’ where each entity is a set of RDF statements serialized in JSON format (cf. Figure 13). In our context, control commands are derived through analyzing the event log by the platform and a FWI is the point where the platform re-generates events with respect to the extracted patterns. Therefore, each function in a FWI (see Figure 14, for an example) corresponds to an event (with same name and signature). We also allow widgets to send their FWI function set along the function signatures to the platform with a handshake process in order to support design-driven and other models. The function signature follows the same semantic approach used for the event descriptions. The communication channel activates a listener in order to receive messages (cf. Figure 12) and it dynamically invokes corresponding target procedures, particularly when control commands are received.


```

1  Functional Widget Interface
2  var DpcPublic = function(){
3      this.dpcSearchWord = DpcSearchWord;
4      ...
5  }
6  function DpcSearchWord(data){
7      ...
8  }
9  ...
10 dpcPublic = new DpcPublic();

```

Figure 14. Functional widget interface of an example widget.

We reflect the aforementioned design to the W3C's widget interface specification as depicted in Figure 15. We define a new attribute named 'channel' for communication, which is made available through initiation of communication channel (cf. Figure 12). An attribute named 'access' is defined for cross-origin communication and is built on top of the communication channel (cf. message type 'access'). The preference feature already exists in the current specification; however, in our case, it is built on top of the communication channel as well (cf. message types 'setPreference' and 'getPreference'). The reason is that the preference and data access requests are handled by the platform.

```

1  Widget Interface
2  interface Widget {
3      readonly attribute DOMString      author;
4      readonly attribute DOMString      description;
5      readonly attribute DOMString      name;
6      readonly attribute DOMString      shortName;
7      readonly attribute DOMString      version;
8      readonly attribute DOMString      id;
9      readonly attribute DOMString      authorEmail;
10     readonly attribute DOMString      authorHref;
11     readonly attribute Storage         preferences;
12     readonly attribute unsigned long height;
13     readonly attribute unsigned long width;
14     readonly attribute Channel         channel;
15     readonly attribute Access         access;
16 };

```

20 Page

Figure 15. Extended W3C widget interface specification.

4.3. User behavior mining and system-driven orchestration

We build our system-driven orchestration approach on two possible conditions (see Figure 16): (1) two or more widgets can consume the same input data, suggesting that these widgets can run in parallel (cf. Figure 5) and (2) one or more widgets can consume the output of another widget, suggesting that the consuming widgets are sequential to the source widget and parallel to each other (cf. Figure 7). In a user-driven orchestration, a user manually provides (i.e., types) the same set (or sub-set) of data to different widgets and manually copies output data from one widget to another. In the approach described herein, the goal is to learn user behavioral patterns, satisfying one of the aforementioned conditions, and appropriate rules, for firing widgets automatically, from the event log.

In the context of this paper, a behavioral pattern is a recurring sequence of user actions connected with control-flow dependencies (e.g., sequence, parallel, choice etc.). Each event

refers to an action and each action refers to a user executable function of a widget. Events are considered as being atomic and associated with data items. We investigate the use of workflow/process mining techniques (van der Aalst et al., 2010) to discover user behavioral patterns from the event logs. A variation of the conventional α -algorithm (van der Aalst et al., 2004) is used to detect patterns and to extract their topologies (i.e., structure). Decision point analysis (e.g., (Rozinat et al., 2008)), is used to find the routing criteria at decision points where routing is deterministic with respect to the value of data attributes in the flow.

Figure 16. Possible scenarios for orchestrating widgets: (1) input-input and (2) output-input.

Workflow mining approaches usually assume that there is a fully connected workflow (process) model to be discovered. In most cases, there is even an a priori prescriptive or descriptive model. In purely event-based approaches, the event log is assumed to be complete (van der Aalst et al., 2004), that is the log is representative and a sufficiently large subset of possible behaviors is observed. The event log is subject to noise due to rare events, missing data, exceptions etc. The use of frequency tables is the most classical approach to deal with the noise (van der Aalst et al., 2004). Existing approaches are focused on the discovery of the topology (i.e., structure), and a few consider how data affects the routing (e.g., (Rozinat et al., 2008; Rozinat and van der Aalst, 2006)) by means of decision mining, yet with a syntactic perspective. In behavior mining, there is not an a priori model. It is very unlikely that there is one single connected workflow, but rather there are small fragments (i.e., patterns) representing commonly repeated user behaviors. In order to emphasize this difference, we call these fragments behavioral patterns rather than models. Completeness (in another form), noise, and effect of data on routing are important issues, which we address through the use of ontologies.

In behavior mining, the completeness problem does not exist as such. In workflow mining, parts of a flow structure, existing in reality, might be missing due to unobserved activities. In behavior mining, on the contrary, the subject user constructs the reality and what exists is what we observe. We consider the completeness problem in a different form however, precisely on the basis of exploration. In order to empower users to explore and utilize the full potential of their environments, recommending possible widgets and actions holds a crucial role. Our semantics-based approach enables such recommendations by matching semantics of inputs (i.e., semantic signatures) and/or outputs (i.e., content) of the widgets. Consider an action named ‘lookforMeaning’ of a widget accepting word type entities and an action ‘searchImage’ of another widget accepting noun type entities, since each noun is also a word (i.e., subclass), one might assert that these two actions can be run in parallel. The same approach is of use to tackle the noise problem. The semantic match between actions can contribute to the frequency analysis as a heuristic factor. The approach also enables us to exploit high-level semantics of the domain in decision point analysis for learning the routing criteria, for instance, by utilizing ontological class types. Assume that a set of consecutive actions are reported along with the syntactic representation of data consumed in each action - see the footprint in situation (1) of Figure 17 for the ‘searchFor’ actions of widgets A, B and C. And assume that this footprint is repeated in the log

substantially, then one might conclude that whenever the 'searchFor' action is executed in widget A by a particular user, consequently, either widget B or widget C is executed (cf. topology). However, it is not possible to learn the routing criteria, since it is indeed dependent on the type of the entity being searched for. In the situation (2) of Figure 17, the event data is enriched with ontological classes, hereby one might conclude that whenever a noun type word is searched in widget A, the widget B follows next, otherwise widget C follows (cf. routing criteria). Although the class type information can be incorporated as a separate syntactic attribute, an ontology-based approach provides reasoning support such as the classification of classes.

```

1  ① Given that:
2  widget_A REPORTS user_1 searchFor STRING 'car'
3  widget_B REPORTS user_1 searchFor STRING 'car'
4  widget_A REPORTS user_1 searchFor STRING 'come'
5  widget_C REPORTS user_1 searchFor STRING 'come'

6  ☛ routing criteria cannot be learned, but one can conclude:
7  IF widget_A.searchFor(input) THEN
8      widget_B.searchFor(input) OR widget_C.searchFor(input)
9      ~~~~

10 ② Given that:
11 widget_A REPORTS user_1 searchFor Noun 'car'
12 widget_B REPORTS user_1 searchFor Noun 'car'
13 widget_A REPORTS user_1 searchFor Verb 'come'
14 widget_C REPORTS user_1 searchFor Verb 'come'

15 ☛ one can conclude:
16 IF widget_A.searchFor(input) THEN
17     IF type_of(input, Noun) THEN
18         widget_B.searchFor(input)
19     ELSE IF type_of(input, Verb) THEN
20         widget_C.searchFor(input)

```

22 Page

Figure 17. Application of ontologies for behavior mining.

Once behavioral patterns are discovered along their topology and routing criteria, the next step is creating a formal representation of these patterns in order to enable validation, verification, sharing, and visualization. For this purpose, we employ Colored Petri nets (cf. (Jensen and Kristensen, 2009; Jensen et al., 2007)) by adopting the approaches presented in Rozinat et al. (2008) and van der Aalst et al. (2004). Petri nets are a graphical and mathematical modeling tool providing the ability to create, simulate, and execute behavioral models; its sound mathematical model allows analysis (e.g., performance), validation, and verification of behavioral models (e.g., deadlock, liveness, reachability, boundness etc.).

A Petri net (see Figure 18) is a graph in which nodes are places (circles) and transitions (rectangles). Places and transitions are connected with directed arcs. Petri nets are marked by placing tokens on places and a transition fires when all its input places (all the places with arcs to a transition) have a token. Colored Petri nets (Jensen and Kristensen, 2009) are a type of high level Petri nets allowing the association of data collections with tokens (i.e., colored/typed tokens). A data value attached to a token is called token color. Next to each place, an inscription determines a set of token colors acceptable for each place (i.e., colored places) and the set of possible token colors specified by means of a type called the color set of a place (see Figure 19). The inscription on the upper side of a place specifies the initial marking of that place (Jensen and Kristensen, 2009). Input arc expressions (cf. Figure 19) can be used to determine when a transition occurs and transition guards can be used to constrain the firing of a transition (cf. Figure 18). In our context, a token corresponds to data

in the flow. All the in/output messages of the widget actions are modeled as colored places, and widget actions themselves are modeled as transitions with input/output places by following adaption of Colored Petri nets for service composition, e.g., (Tan et al., 2010). Learned routing criteria can be represented in terms of rules in the form of arc expressions or guards. The work presented in Gasevic and Devedzic (2006) introduces a Petri net ontology, and can be integrated to utilize ontological reasoning and to support the sharing of behavioral patterns (also see Vidal et al. (2010)).

Figure 18. Petri nets for widget orchestration: transitions refer to widget functions.

The specifics of our approach are presented in the following. Note that the goal is not to mine full behavioral models, but only the fragments of them, and to realize automated orchestration through the extracted fragments. These fragments need to be simple, yet sound, and short in depth. This is because the target of automation is the end-users, and it is not realistic to bring an excessive number of automated actions to a user's attention.

Figure 19. The patterns used in automated orchestration: OR and Sequence.

Hereby, in our context, a pattern consists of a triggering action and one or at most two parallel/alternative consequent actions. If there is a single follower action, it is sequential to the triggering action in terms of the execution order. If there are two follower actions, these are sequential to the triggering action in terms of the execution order, and are either parallel to each other or there is a choice in between. The choice between two alternative actions might dependent on constraints (e.g., one widget cannot process a particular type of data attribute) or on the preferences of the end-user. Therefore, we limit our approach to two types of patterns (cf. Figure 19): (1) Multi-Choice pattern (a.k.a. OR-split) and (2) Sequence pattern (cf. (Mulyar and van der Aalst, 2005)). The Multi-Choice pattern allows execution of one or more transitions out of several available transitions. The realization of the Multi-Choice pattern is depicted at the left-hand side of the Figure 19 (e.g., B^{f1} and C^{f2}, only C^{f2}, or only B^{f1} – the main letter denotes the widget and the superscript denotes the widget

function), and is based on input arc expressions where conditions are either mutually exclusive or overlapping (i.e., combined representation of AND-split, XOR-split and OR-split). The Sequence pattern allows execution of a single action after the triggering action without any alternative. In this respect, the goal becomes, for each possible action a , to detect the most frequent two actions that can run upon the execution of the action a and to extract the decision criteria if there exists a choice in between the selected two actions.

Regarding the pattern extraction procedure, an event log is the starting point. In workflow mining, a log consists of a set of traces and each trace consists of an ordered set of events (i.e., with respect to associated timestamps). Each trace in a log corresponds to a workflow instance (a.k.a. case), and reveals a possible execution path of the target workflow model. Therefore, given a set of traces, which is a sufficiently large subset of possible behaviors, it becomes possible to derive a complete workflow model. However, in our context, the original log indeed consists of a set of user sessions and in every session different pattern instances coexist including arbitrary actions in the form of a continuous series of events. For this reason, the log needs to be processed to generate a meaningful set of traces for each action. Input-input match based patterns and output-input match based patterns are processed separately to extract traces for the sake of simplicity. For the former, we split the log into a set of fragments for every possible triggering action a . More specifically, fragments for action a are constructed by taking each occurrence of event a in the log along z number of predecessor events and z number of successor events (z is the window size that can be defined with respect to the total number of actions, $z=2$ in our experiments). Data associated with each event in a fragment is matched with data associated with action a where an event should consume the same or a subset of the triggering action a 's data. Typed entities are compared with the subsumption test where a subclass relationship should hold (either direction). Events that do not match, any repetitive events, and any re-occurrence of event a are removed from the corresponding fragment. Each resulting fragment represents a trace for action a . Traces based on output-input matches are special, and they are based on specific events: 'dataSelected' (i.e., copy) and 'formSelected' (i.e., paste) (cf. Section 4.1). This is because these two events are one of the best means to capture patterns based on output-input matches. Considering that output of a widget is mostly not a single data chunk, rather a set of data chunks, an output-input match detection would only be possible through comparing all output data of a widget with the candidate widget's input attributes. However, the end-user data mobility facility does not only enhance the user experience but also allows us to detect output-input matches by enabling us to identify the user selected data chunk. The resulting traces are in the form of a series of events consuming the same set (or subset) of data for the input-input match based traces and in the form of series of paste events following a single copy event (e.g., 'dataSelected', 'formSelected', 'formSelected'...) for output-input match based traces. The trace extraction process also eliminates the noise emerging from the arbitrary user events.

After the trace extraction process, the task is to identify the most frequent two follower events for each action, hence actions associated with these events. For this purpose, we employ a substantial variation of the frequency analysis used for the α -algorithm (van der Aalst et al., 2003). Let W_a be a trace set, extracted from the user log W for the action a , over the set of actions L . Let $a, b \in L$; $a \ggg_w b$ if and only if there is a trace $\sigma_a = t_1 t_2 t_3 \dots t_{n-1}$, $i \in \{1, \dots, n-2\}$, and $j \in \{2, \dots, n-1\}$ such that $\sigma_a \in W_a$, $t_i = a$, $t_j = b$, and $i < j$. $a \lll_w b$ if and only if there is a trace $\sigma_a = t_1 t_2 t_3 \dots t_{n-1}$, $i \in \{1, \dots, n-2\}$, and $j \in \{2, \dots, n-1\}$ such that $\sigma_a \in W_a$, $t_i = b$, $t_j = a$, and $i < j$. A metric indicating the strength of the frequency between action a and action b (denoted with $\#A \rightarrow B$), where a is the triggering action and either $a \ggg_w b$ or $a \lll_w b$ holds, is calculated as follows. For the input-input based traces, if action a occurs before action b ($a \ggg_w b$) or action b occurs before action a ($a \lll_w b$) in a trace and n is the number of intermediary events between them, $\#A \rightarrow B$ frequency counter incremented with a factor of δ^n (δ is frequency fall factor $\delta \in [0.0 \dots 1.0]$, in our experiments $\delta = 0.8$). The contribution to the frequency counter is maximal 1, if action b occurs directly after or before action a ($n=0$ and

$\delta=1$). For the output-input based traces only the $a \ggg_w b$ relation exists where a is the triggering copy action. After processing the whole trace set of action a , the frequency counter is divided by overall frequency of the action a denoted by $\#A$. This process is repeated for every action over its associated trace set. For each triggering action a , two follower actions, with the highest frequency factor above a specified threshold th , is selected (in our experiments $th=0.45$). The value of th can be adjusted with respect to the total number of actions or can be set for each triggering action individually (possibly with respect to the number of potential follower actions). It is also possible that no action or only one action is above the threshold. For the former, no follower action is selected and for the latter the pattern becomes a sequence in terms of the execution order (i.e., in a input-input based pattern, theoretically a follower action is parallel to the triggering action; however, it is sequential to the triggering action in terms of the execution order since it is only executed once the triggering action occurs).

For every action a , for which at least one follower action is selected, a decision point analysis (cf. (Rozinat et al., 2008)) is conducted for determining the firing conditions. Following this idea, every decision point becomes a classification problem, that is classifying a given event instance into a class, representing a follower action with respect to previous observations (i.e., training data). However, the decision point analysis employed by Rozinat et al. (2008) does not consider Multi-Choice patterns (only XOR – a.k.a. Exclusive Choice); hence, the authors approaches the case as a single label classification problem. We consider the problem as a multi-label classification problem where multiple target class labels can be assigned to an instance. Multi-label classification problems can be handled through problem transformation or algorithm adaptation methods. Problem transformation methods force the learning problem into the traditional single-label classification where algorithm adaptation methods adapt an algorithm to directly perform multi-label classification (Tsoumakas and Katakis, 2007).

Figure 20. Multi-label classification – the resulting decision tree is transformed into a rule set.

We follow the problem adaptation method, when more than one follower action is selected, and apply a well known data mining algorithm, namely C4.5, for the classification. C4.5 (Quinlan, 1993) is used to generate decision trees. In the single label classification, a being the triggering action and b, c being the follower actions, an event instance a can be classified into either only b or only c (that is either action b or c follows); however, with the multi-label classification, an event can be classified into only b , only c , or b and c . The approach we use, for the problem transformation, is a variation of the label combination

approach (Boutell et al., 2004), which considers events with multiple labels as a new class (e.g., b , c , and $b-c$). We also add a fourth class named *rest* referring to negative examples that are not associated either with b or with c . An example training data set, its classification, and rule generation are exemplified in Figure 20 for the triggering action a and selected follower actions b and c . In the training data set (cf. left-hand side of the Figure 20), each line corresponds to an event instance associated with action a , and each column represents a data attribute. The final column represents the action that followed this event instance. If an event a is followed by both action b and c in the corresponding trace (directly or indirectly), the class label for this event instance is set to the combination of labels of follower action classes (cf. Figure 20). Given a training data, a C4.5 classifier is trained to classify unlabeled instances. Then, the generated decision tree is transformed into a rule set (cf. the right hand side of the Figure 20). Each path from the root node of the decision tree to a leaf node represents a different rule. Note that if a single follower action is selected, we conduct a binary classification with C4.5.

The obtained rules are committed to the run-time platform (cf. Section 4.2). Whenever an event occurs, the associated event data is compared with the antecedents of the rules available in the rule set of the corresponding event. If a rule is satisfied, the action it specifies in its consequent is executed through submitting a control command to the corresponding widget. If the matching rule has a single label as its consequent, other rules in the rule set are also checked. If the matching rule has a multi-label consequent, the remaining rules are not checked. Regarding the subsumption checking for the entity type constrains taking place in the rule antecedents, dedicated approaches such as class hierarchy encoding technique described in Preuveneers and Berbers (2008) can enhance the client-side performance.

5. Evaluation and Discussion

We have implemented a partial prototype to prove the applicability of our approaches. Several widgets have been developed for language learning, by adopting the W3C widget specification, with respect to the extensions that we propose; hence, a widget-based personal learning environment (WIPLE) is realized. The widgets we have developed are cross-platform (e.g., Opera platform - <https://widgets.opera.com> – enables widgets to run on desktop). A demo can be watched online from the following web address <http://www.ahmetsoylu.com/pubshare/program2012/>.

An example scenario is depicted in Figure 21 where a user watches a video material in the ‘mediatic’ widget and clicks on a particular word in the subtitle. For this particular user and widget action, the mostly used follower actions belong to the ‘dafles’ widget and the ‘dpc’ widget (i.e., input-input match). However, there is a choice in between ‘dafles’ and ‘dpc’ with respect to certain rules (cf. (1) in Figure 21). If the clicked word is in English, only ‘dpc’ follows (‘dpc’ accepts words in English, French, and Dutch whereas ‘dafles’ is a French dictionary accepting only French words); if the word is in French and of the verb type, ‘dafles’ follows; and if the word is a French noun, both widgets follow. Assume that the word the user clicked on is a French noun, e.g., ‘voiture’ (car); in that case, both ‘dafles’ and ‘dpc’ widgets are automatically executed. Afterwards, the user clicks on an output item of the ‘dafles’ widget. For the ‘dafles’ widget, there is only one follower action which belongs to the ‘flickr’ widget (i.e., output-input match) with a condition constraining the type of selected output entity to a noun (cf. (2) of Figure 21). Since the ‘flickr’ widget is not at the near vicinity of the ‘dafles’ widget, first it is automatically moved closer to the ‘dafles’ widget (cf. (Gali and Indurkha, 2010) and (3) of Figure 21). Then, the selected output data chunk is copied to the target ‘flickr’ HTML form (note that the user does not need to click on the marker of the target form). At this point, the user activates the ‘flickr’ widget (i.e., clicks on ‘zoek’ (search) button), and relevant images are retrieved and displayed (cf. (4) of Figure 21).

The orchestration process is semi-automatic because, for output-input matches, the end-user needs to activate the target widget after data is copied. Indeed, this episode can be fully automated or the same approach can be applied for input-input matches (i.e., copying input data to the target widgets without activating the widget action). Regarding the former, it must be checked that the copied data is sufficient to execute the corresponding actions. However, if the target actions realize sensitive operations such as insert, delete etc. over data, the latter approach might be more appropriate. Application of both approaches is independent of the platform, and depends on the FWIs of individual widgets. Note that semi-automatic widget orchestration is not considered to be hybrid orchestration since the latter requires the involvement of users while selecting appropriate actions to execute.

Figure 21. An example scenario for system-driven widget orchestration.

Regarding the graphical environment, each widget container is associated with a set of presence related operations (cf. (5) of Figure 21) such as move, minimize, close, pin, settings etc., and the corresponding visual elements appear when the user moves the cursor over a widget. There exists a task bar where widgets can be minimized to (cf. (6) of Figure 21). The task bar also includes a 'Widget Store' tab for adding new widgets, repositories etc. Different workspaces can be created and accessed through the task bar; an alternative can be the use of the browser tabs that will allow a more natural access to different workspaces. Each workspace can be accessed through a distinct URL over different browser tabs.

5.1. Qualities of the approach

There exist several approaches for widget orchestration. Prominent ones, namely user-driven, design-driven, distributed, system-driven, and hybrid approaches, have been introduced in Section 2.2 and Section 3. A qualitative comparison of these approaches is given in Table 1 in terms of several major interlinked properties.

An end-user driven approach (without any end-user data mobility support) is fully demand-driven since the control of orchestration is totally held by the end-users where design-driven and distributed approaches lack this characteristic, since the control logic depends on the perception of the skilled users or programmers. System-driven and hybrid approaches maintain the demand-driven characteristic implicitly because the automation logic is extracted from the end-user logs.

Table 1. Comparison of personal environments based on different orchestration approaches.

	User-driven	Design-driven	Distributed	System-driven	Hybrid
Demand-driven	+++++	-	-	++++	++++
Open	+++++	++	+	++++	++++
Loosely coupled	-	+++	+++++	++++	++++
Clustered	+++++	+++++	+++++	+++++	+++++
Simple (Widget Dev.)	+++++	++++	++	++++	++++
Effortless (Orch.)	+	+++++	+++++	+++++	++++
Sound (Orchestration)	+++++	++	+	+++	++++
Autonomous (Orch.)	-	-	+++++	+++++	++++

We consider openness in terms of (1) platform and (2) end-user experience. We evaluate openness of the platform in two folds, (a) end-user's ability to add new widgets to her environment and (b) entry barriers (i.e., commitments) that widget/component providers need to overcome. We evaluate openness of the end-user experience as the level of freedom that an end-user has in the orchestration process. A user-driven approach is expected to exhibit a high level of openness since lesser technical commitments and obligations are required (to standards and specifications) and the experience is driven by the end-user herself. A design-driven approach is weak regarding the openness of the end-user experience since the experience is mostly pre-designed. In terms of the platform, it requires more, albeit not substantial, commitments. A distributed approach is similar to a design-driven approach with respect to the openness of the end-user experience; however, it is ranked lower in terms of openness of the platform since each widget also has to implement its own orchestration logic. System-driven and hybrid approaches are more open compared to other approaches, excluding the user-driven orchestration, in both perspectives. This is due to the fact that the end-user experience is driven by the system with implicit user control. In terms of the platform, they require more, albeit not substantial, commitments.

28 Page

Loose coupling can be evaluated at several levels, such as physical coupling, communication style, type system, interaction pattern, control of process logic, service discovery and binding, and platform dependencies, as described in Krafzig et al. (2004). For a user-driven approach, we consider loose coupling as a non-existing property because there is no explicit interaction (e.g., data exchange, message communication etc.) between widgets in a user-driven scenario. The level of coupling in other approaches might vary depending on the underlying implementation approach; however, most of the approaches presented in Section 3 follow some blueprints to achieve loose coupling. More specifically, there is no direct physical link between widgets, the communication style is mainly asynchronous, payload semantics is employed rather than strongly typed interface semantics, a data centric interaction pattern is used etc. The system-driven and hybrid approaches are ranked slightly lower due to the central control logic (which is indeed mined, yet in short-term, these approaches are vulnerable to the structural changes in event payloads and if the central control logic fails the whole system fails). The design-driven approach is ranked lowest mainly due to the static functional binding and highly centralized control logic. All the approaches are based on the existence of standardized widget engines guaranteeing platform and OS independence.

The simplicity of the widget development is indeed dependent on the amount and complexity of the required commitments. A user-driven approach can be ranked higher in this respect since the required commitments are minimal. Except the distributed approach, the remaining approaches can be ranked slightly lower due to higher commitments. A distributed approach is ranked lowest since each widget also has to implement its own logic of responsiveness. This becomes particularly difficult due to synchronization problems (i.e., widgets are independent – cf. Section 3). Note that the commitment to standards and specifications is inevitable primarily due to interoperability considerations; however, the

more complex and voluminous a standard/specification, the more difficult it will produce simple and open approaches. Therefore a minimalist approach is required.

We consider effortlessness of orchestration from the end-user perspective (i.e., the level of explicit user effort required for the orchestration). A user-driven approach requires end-users to drive their own experience manually where in other approaches orchestration takes place automatically. Therefore one can consider orchestration process in other approaches easier to be realized compared to a user-driven approach (neglecting the appropriateness of the automated actions for the moment). However, in a hybrid approach, control is shared between the end-user and the system; therefore, a hybrid approach is ranked slightly lower compared to other approaches.

The soundness of orchestration refers to the appropriateness of the automated actions in an orchestration process. A user-driven approach can be ranked highest, since it is the end-user who directly chooses the next action. A design-driven approach and distributed approach is ranked lowest among others because the end-user has no implicit or explicit control in the orchestration process. A design-driven approach is ranked slightly higher compared to a distributed approach, since at least a programmer or skilled user (i.e., a human actor) evaluates possible scenarios and designs the control logic accordingly. System-driven and hybrid approaches are based on implicit user control (cf. behavioral patterns); therefore, they can be ranked higher compared to design-driven and distributed approaches. Yet, compared to a user-driven approach, they are ranked lower, since mined patterns cannot be considered equal to the explicit user request in terms of their reliability and accuracy. A hybrid approach is slightly better than a system-driven approach, since control is shared between the end-user and the system and at certain points explicit user input is requested.

Autonomy of orchestration is regarded as the capacity of an orchestration process to be self-organized, that is its control logic is not explicitly driven by an end-user or programmer. This does not hold for user-driven and design-driven approaches. Other approaches are not directly dependent on a human user for the orchestration logic. A hybrid approach is ranked slightly lower since at certain points it requires human involvement.

Given the aforementioned comparisons and discussion, system-driven and hybrid approaches are ranked highest in overall. There exist a trade of between two approaches in terms of soundness and simplicity of the orchestration. Since a hybrid approach mediates with the end-user at some points (e.g., in a case of uncertainty), the resulting automation can be more reliable. However, control is shifted towards the end-user. Note that user involvement (cf. (Soylu et al., 2011a; Spiekermann, 2008)) in automation is required at some degree, particularly under ambiguity and when severe implications are probable (e.g., delete, insert actions etc.). A hybrid approach compromises autonomy and easiness of orchestration (from end-user point of view) for the soundness of orchestration.

5.2. Patterns and decision points

One of the problems with Petri nets and some of the process mining techniques is the representation and the identification of advanced patterns, for our context, particularly Multi-Choice patterns; the α -algorithm can only mine some kind of OR patterns while there are other algorithms, such as a multi-phase miner and region based techniques, that can mine larger classes of process models (van der Aalst, 2011a; b). Nevertheless, Multi-Choice patterns even might be reduced to XOR-splits or AND-splits by most of the noise elimination techniques. In our approach, we adopted frequency analysis (that is normally used for noise elimination) to detect the mostly appearing follower actions. The topology and the routing criteria are identified with decision point analysis. Therefore, the limitations regarding Multi-Choice pattern do not apply to our approach. Although Petri nets do not provide any means for explicit representation of OR-splits, since patterns subject to our work are small scale

(i.e., at most 3 transitions), the Petri net representation based on AND-split and arc expressions are sufficient (cf. Figure 19).

Probably due to the aforementioned considerations, the decision mining approach employed for decision point analysis in Rozinat et al. (2008) and Rozinat and van der Aalst (2006) omits OR-splits (i.e., Multi-Choice). The label combination approach that we follow, for multi-label classification, might be problematic since it may lead to data sets with a large number of classes and few examples per class (Tsoumakas and Katakis, 2007). Let L be the set of disjoint labels; the number of class labels in a combined label set L_c is the sum of k -combinations of L for every possible value of k where $1 \leq k \leq |L|$. That is combined label set L_c increases in polynomial magnitude with respect to $|L|$:

$$|L_c| = \sum_{k=1}^{|L|} C(|L|, k)$$

However, this does not affect our approach due to the small size of L , i.e., $|L|=2$ and $|L_c|=3$. The affect can be analyzed through two concepts, namely label cardinality and label density, introduced in Tsoumakas and Katakis (2007). Let D be a multi-label data set with $|D|$ number of multi-label examples (x_i, Y_i) , $i=1 \dots |D|$ where x_i refers to a particular instance and Y_i refers labels associated with x_i . The label cardinality and label density of D is defined as follows:

$$LC(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} |Y_i| \text{ and } LD(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i|}{|L|}$$

The label cardinality of D is the average number of labels of the examples in D , and the label density of D is the average number of labels of the examples in D divided by $|L|$ (Tsoumakas and Katakis, 2007). Our case is a 2-class multi-label classification problem where $|L|=2$. Each example in D is associated with at least one label. Therefore, in one extreme (the target pattern is an XOR-split), every example will be associated with one label which will make $LC(D)=1$ and $LD(D)=0.5$, and in the other extreme (the target pattern is an AND-split), every example will be associated with two labels which will result in $LC(D)=2$ and $LD(D)=1$. Accordingly, in our context, the label density is minimum 0.5. We compare this minimum label density with the densities of twenty-two open-access multi-label data sets ($|L|>2$), from diverse domains, which are available at <http://mulan.sourceforge.net/datasets.html> (Mulan (Tsoumakas et al., 2011) - an open-source Java library for learning from multi-label datasets). Information regarding the data sets and comparison results are shown in Table 2. The final column (2C ratio) presents 2-class minimum density ratio to the individual densities of the data sets. According to the results, after excluding the extreme cases, EUR-Lex (ed) and EUR-Lex (dc) data sets, the average ratio is 19,8 (i.e., on average a 2-class multi-label data set is 19.8 times denser than a multi-label data set with $|L|>2$). The results confirm that, in our context, any 2-class multi-label data set will probably be denser than other multi-label data sets with higher number of labels. This also holds in terms of attribute numbers where a large number of attributes is another reason for sparse data sets. In the data sets presented in Table 2, the number of attributes varies from min. 72 to max. 49060; a widget action is not expected to consume more than 6-7 attributes. Therefore we expect that the approach will not encounter a severe sparse data set problem.

If one wants to perform multi-label classification with a higher number of labels, e.g., where more than two follower actions are necessary, the use of algorithm adaptation methods is preferable; the Mulan library (Tsoumakas et al., 2011) includes a variety of state-of-the-art algorithms for performing several multi-label learning tasks.

Table 2. Density comparison of multi-label data sets in which $|L|>2$ with 2-class multi-label data sets in which $|L|=2$ (min. density 0.5).

Data set	Instance #	Attr. #	Label #	Cardinality LC(D)	Density LD(D)	Ratio 0.5/LD(D)
bibtex	7395	1836	159	2.402	0.015	33.3
bookmarks	87856	2150	208	2.028	0.010	50.0
CAL500	502	68	174	26.044	0.150	3.3
corel5k	5000	499	374	3.522	0.009	55.6
corel16k	13811	500	161	2.867	0.018	27.8
delicious	16105	500	983	19.020	0.019	26.3
emotions	593	72	6	1.869	0.311	1.6
enron	1702	1001	53	3.378	0.064	7.8
EUR-Lex (dc)	19348	5000	412	1.292	0.003	166.7
EUR-Lex (sm)	19348	5000	201	2.213	0.011	45.5
EUR-Lex (ed)	19348	5000	3993	5.310	0.001	500.0
genbase	662	1186	27	1.252	0.046	10.9
mediamill	43907	120	101	4.376	0.043	11.6
medical	978	1449	45	1.245	0.028	17.9
rcv1v2 (subset1)	6000	47236	101	2.880	0.029	17.2
rcv1v2 (subset2)	6000	47236	101	2.634	0.026	19.2
rcv1v2 (subset3)	6000	47236	101	2.614	0.026	19.2
rcv1v2 (subset4)	6000	47229	101	2.484	0.025	20.0
rcv1v2 (subset5)	6000	47235	101	2.642	0.026	19.2
scene	2407	294	6	1.074	0.179	2.8
tmc2007	28596	49060	22	2.158	0.098	5.1
yeast	2417	103	14	4.237	0.303	1.7

At the moment we use an offline learning approach, that is, after collecting a substantial amount of data, patterns and decision rules are generated. However, we face a concept drift problem (cf. (Widmer and Kubat, 1996)) since the distribution underling the instances or the rules underlying their labeling might change over time (e.g., user’s preferences change). This is because, after certain patterns and rules are learned and put into effect, the events and data occurring later will be the result of the automated actions themselves and changes in user preferences and the environment will not be reflected. Therefore, it is a must to develop methods and techniques to alleviate this problem and probably to enable end-users to communicate inappropriate automations without disrupting the end-user experience by putting end-users under an excessive load. We are also interested in the possibility to generate patterns and rules with on-line learning through data stream mining (cf. (Gaber et al., 2005)) by extracting patterns from continuous data records. There already exist algorithms and frameworks to support data stream mining (e.g., (Bifet et al., 2010)).

5.3. End-user experiment and assessment

We conducted a preliminary user experiment and survey in order to test the effectiveness of our approach. The user experiment and survey were performed with four widgets and six users. The profiles of the test users are given in Table 3 (1 for ‘very poor’, 5 for ‘very good’).

The first goal of the user experiment was to test the performance of the mining approach. The experiment was realized in three sessions for each user individually (1.5 hours in total for each user). A five minutes introduction to WIPLE and widgets was given to each user, and users were given the opportunity to get familiar with the WIPLE and widgets before the experiment. The first session consisted of four cycles; at each cycle, the users were given fifteen English words, and asked to comprehend the word by using the existing widgets (i.e., the ‘flickr’ widget that retrieves images associated with a particular word, the ‘mediatic’ widget that allows watching videos with subtitles, the ‘dpc’ widget that retrieves example sentences which include a specified word, and a new widget named ‘engDict’ that is

an English-to-English dictionary). However, each cycle had a specified widget that the user had to start off with for each word (e.g., for cycle 2, the user always had to use the 'DPC' widget first, for each word, before using any other widgets) in order to ensure even data distribution. A total of sixty English words were used at the first session. The second session had four cycles as well and was similar to the first session; however, only ten words were given per cycle. A total of forty words were used in the second session. The words, which were used at the first and second sessions, were selected at a difficulty level above the ability level of the test users in order to ensure use of multiple widgets for each case (i.e., word). The first session was used for generating training data regarding the usage behaviors of the test users. After the first session, patterns were mined for each user. The data gathered at the second session was used as test data.

Table 3. Profiles of the test users.

User	Occupation	Age group	# of years using internet	Frequency of internet use	Familiarity with mashups
User 1	Teacher	20-25	7	daily	2
User 2	Engineer	25-30	12	daily	3
User 3	Teacher	30-35	14	daily	1
User 4	Student	20-25	8	daily	1
User 5	Student	20-25	7	daily	2
User 6	Student	15-20	4	daily	1

Recalling our approach, at the first stage, we select the mostly used two following actions for each action and perform a 2-class multi-label classification to mine the selection criteria at the second stage. We have analyzed the second stage in Section 5.2 and we want to evaluate the overall two-stage approach here. The overall approach itself can be considered as a multi-label classification problem as well where L is the set of widget actions excluding the triggering action itself which is subject of the classification. The L should not be confused with the label set used in 2-class multi-label classification at stage two which is simply the subset of L denoted here. In this respect, we can use evaluation metrics used for multi-label classification problems to evaluate the overall approach. In multi-label classification a result can be fully correct, partly correct, or fully incorrect (Boutell et al., 2004). For instance, say 'engDict' is found to be most frequent for the 'mediatic' widget along with an empty decision criterion (i.e., follows in any case) for a particular user. In the test session, after using the 'mediatic' widget, the test user might execute 'engDict' (fully correct), 'dpc' (fully incorrect), 'engDict' and 'dpc' (partly correct), 'engDict' and 'flickr' (partly correct) etc. The following metrics (Tsoumakas and Katakis, 2007) were used for performance evaluation: (1) Hamming Loss (Schapire and Singer, 2000), (2) accuracy, (3) precision, and (4) recall (Godbole and Sarawagi, 2004).

Let D be a multi-label test data set with $|D|$ multi-label examples (x_i, Y_i) where $i = 1 \dots |D|$ and Y_i subset of L ; let H be a multi-label classifier and $Z_i = H(x_i)$ be the set of labels predicted by H for x_i (Tsoumakas and Katakis, 2007). Accordingly, Hamming Loss, accuracy, precision, and recall are defined in the following where Δ stands for the symmetric difference of two sets, m is the total number of widget actions, and $|L| = m-1$.

$$\text{HammingLoss}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \Delta Z_i|}{|L|}, \text{Accuracy}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

$$\text{Precision}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \cap Z_i|}{|Z_i|}, \text{and Recall}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \cap Z_i|}{|Y_i|}$$

The evaluation results are shown in Table 4. Although the approach is yet to be experimented with larger user groups and a higher number of widgets in different contexts, the experiments resulted in extraction of different patterns due to varying characteristics of the test users; therefore the results suggest that the mining approach is indeed promising. The user experience with the automated system (the third session which is described in what follows) revealed that indeed the accuracy, precision, and recall can deviate from what we have observed. This is due to the possibility that users might feel uncomfortable or comfortable with an automated action whether it does or does not follow his intention. Therefore, in future experiments we plan to measure perceived accuracy, precision and recall along their observed values. In order to acquire perceived values, another controlled session with automation can be conducted. After each automation users can be asked to comment if they see the automated widgets appropriate and if not which widgets they were about to use.

Table 4. Analysis of the test results.

Metric	User 1	User 2	User 3	User 4	User 5	User 6
HammLoss	0.20	0.15	0.15	0.13	0.13	0.19
Accuracy	0.72	0.77	0.80	0.80	0.78	0.72
Precision	0.77	0.86	0.90	0.80	0.81	0.80
Recall	0.83	0.91	0.81	0.80	0.91	0.76

The third session of the experiment was used as a basis for the end-user assessments and the usability analysis by means of an experience survey, and realized through a think-aloud manner. In Virzi (1992), the author suggests that five-six test users are usually sufficient to find major usability problems (around %85); therefore, the aim of this preliminary user study was to detect major usability problems in early stages of our prototype and to provide input for the ongoing development process. In this third session, the generated rules were put in force in the platform and users were asked to use the system again with a different set of words. During the session, users were asked five Likert-scale questions: (1) ‘How useful do you find the mashup idea regardless of the prototype that you have just used?’, (2) ‘How successful do you find the system you have just used?’, (3) ‘How do you like the data mobility facility?’ , (4) ‘How do you like the orchestration facility?’, and (5) ‘How do you like the dynamic widget relocation?’. Users were first asked to comment on the question including any recommendations and then to give a rank between 1 (very poor) and 5 (very good). The survey results are shown in Table 5.

Table 5. End-user survey results.

Concept	User 1	User 2	User 3	User 4	User 5	User 6
(1) Mashups	5	5	5	5	5	5
(2) WIPLE	3	3	4	3	4	4
(3) Data mobility	5	5	5	4	4	5
(4) Orchestration	5	5	5	5	5	5
(5) Relocation	4	4	5	3	5	5

Test users arguably found the widget-based mashup idea quite useful and promising; most of the users immediately commented on possible uses and scenarios. Regarding our WIPLE implementation, expectedly, users found it yet to be improved. Users mainly demanded a more uniform interface and a higher degree of customization such as widget sizes, colors etc. Users mainly found the data mobility facility useful; two users commented that rather than using mouse clicks for copying data, they would prefer a drag and drop

facility supported with more visual cues (e.g., an animation). The widget orchestration facility was found very useful and users mostly reported on their past experiences in which they had needed such a facility. Widget relocation was mainly found useful; however, two users did not prefer sliding effect since they found it time consuming. One user commented that this could be customized (e.g., glow effect, sliding etc.) and user preferences on widget locations could also be learned and reflected while moving interacting widgets closer. The results suggest that the approach does not include any major usability problems; however, the platform interface has to be improved along with a customization support.

During the first and second session, we observed that there might be various factors, other than the match between widgets, affecting the preferences of a user while combining functionalities of different widgets. For instance, two users, with comparatively lower language levels, usually omitted the 'dpc' widget, since they did find the example sentences provided by 'DPC' widget pretty long and complex. These two users were also indifferent to the word types and firstly preferred checking the meaning of the word and then possibly its image if it is a concrete noun. Another user with comparatively higher level language skills usually differentiated nouns and verbs and preferred checking the image of the word more often; however, the user found most of the retrieved images irrelevant and did not visit this widget as much as expected. These observations confirm that an orchestration approach, that is only based on syntactic or semantic match between widgets, is far from satisfying user preferences; however, the match between widgets can be quite useful to guide naive end-users to explore different possibilities.

6. Conclusion and Future Work

In this paper, we first have provided a broad overview of the mashup landscape with a holistic approach and set the links between widget-based UI mashups with orchestration, which we named mashups by orchestration, and web-based personal environments. We have discussed several prominent approaches for the realization of mashups by orchestration and opted for a system-driven approach where the system (i.e., the widget platform) learns user behavioral patterns from the user logs and automates the interplay between widgets accordingly. We have identified several generic challenges concerning most of the possible approaches and specific challenges mainly concerning a system-driven orchestration approach. We have described and addressed these changes in three main folds: (1) widget interoperability, (2) platform and architecture, and (3) user behavior mining. We have investigated widget interoperability in two levels, namely application interoperability and data interoperability. We have introduced functional widget interfaces (FWI), with which widgets disclose their functionality, for application interoperability and employed semantic web technologies, particularly ontologies and embedded semantics (e.g., eRDF, RDFa, microformats, microdata), for data interoperability. We have built an end-user data mobility facility on top of this interoperability infrastructure; this facility allows end-users to copy data from one widget to another. For this purpose, we have specified techniques for annotating HTML forms and matching user selected data with the HTML form elements. We have specified a reference platform and architecture, introduced a communication channel and message format, and extended W3C's widget specification respectively. We have employed workflow mining and multi-label classification techniques within a two-stage approach for mining behavioral patterns in terms of their topology and routing criteria. We have employed Colored Petri nets for representation of the behavioral patterns. Finally, we have provided a preliminary evaluation of our approach in terms of its qualities, performance of the mining approach, and its usability with a prototype. The results suggested that our approach is promising.

The future work firstly includes maturation of the prototype and larger scale experiments for different application domains. Investigation of methods and techniques for dealing with concept drift problem and the exploration of applicability of an on-line pattern

mining approach based on data stream mining are also of crucial. We believe that with the standardization of widget technologies, e.g., widgets, platforms (e.g., run-time system/environment, development frameworks etc.), and reference architectures, the widgetization of existing web applications (particularly dynamic approaches) will be of crucial importance due to the growing interest for personal environments and widget-based UI mashups in different domains (e.g., (Back and Bailey, 2010)). In this respect, we are interested in designing means for the automated widgetization of existing applications through harvesting application semantics from the interfaces of applications (i.e., with embedded semantics, cf. (Soylu et al., 2012)). The interface annotation can be automated if the original application is developed with a Model Driven Development approach employing ontologies as a modeling paradigm (cf. (Fraternali et al., 2010; Soylu et al., 2011a)). Finally, an interesting application of widget-based UI mashups is to enable end-users to program their personal and pervasive environments in which digital and physical entities are encapsulated by means of widgets (cf. (Kindberg et al., 2002; Laga et al., 2010; Soylu et al., 2010a)). In other words, the aim is to allow end-users to generate mashups by composition through demonstration to program their pervasive spaces (cf. (Srblic et al., 2009; Tuchinda et al., 2011)). Our infrastructure and learning approach can be adapted for such purposes.

Acknowledgements

This article is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF Knowledge platform ‘Harnessing collective intelligence in order to make e-learning environments adaptive’ (IOF KP/07/006). Partially, it is also funded by the European Community's 7th Framework Programme (IST-FP7) under grant agreement no 231396 (ROLE project).

References

Adida, B. (2008), "hGRDDL: Bridging microformats and RDFa", *Journal of Web Semantics*, Vol. 6, No. 1, pp. 54-60.

Ankolekar, A., Krotzsch, M., Tran, T. and Vrandecic, D. (2008), "The two cultures: Mashing up Web 2.0 and the Semantic Web", *Journal of Web Semantics*, Vol. 6, No. 1, pp. 70-75.

Back, G. and Bailey, A. (2010), "Web Services and Widgets for Library Information Systems", *Information Technology and Libraries*, Vol. 29, No. 2, pp. 76-86.

Baresi, L. and Guinea, S. (2010), "Consumer Mashups with Mashlight", in *Towards a Service-Based Internet (ServiceWave 2010) proceedings of the third european conference in Ghent, Belgium, 2010*, Springer-Verlag, Berlin, pp. 112-123.

Benslimane, D., Dustdar, S. and Sheth, A. (2008), "Services mashups - The new generation of web applications", *IEEE Internet Computing*, Vol. 12, No. 5, pp. 13-15.

Bettini, C., Brdiczka, O., Henriksen, K., Indulska, J., Nicklas, D., Ranganathan, A. and Riboni, D. (2010), "A survey of context modelling and reasoning techniques", *Pervasive and Mobile Computing*, Vol. 6, No. 2, pp. 161-180.

Bifet, A., Holmes, G., Kirkby, R. and Pfahringer, B. (2010), "MOA: Massive Online Analysis", *Journal of Machine Learning Research*, Vol. 11, pp. 1601-1604.

Bizer, C., Heath, T. and Berners-Lee, T. (2009), "Linked Data - The Story So Far", *International Journal on Semantic Web and Information Systems*, Vol. 5, No. 3, pp. 1-22.

Boutell, M. R., Luo, J. B., Shen, X. P. and Brown, C. M. (2004), "Learning multi-label scene classification", *Pattern Recognition*, Vol. 37, No. 9, pp. 1757-1771.

Brusilovsky, P. and Maybury, M. T. (2002), "From adaptive hypermedia to the adaptive web", *Communications of the ACM*, Vol. 45, No. 5, pp. 30-33.

Cáceres, M. (ed.) (2011), "Widget Interface", W3C, available at: <http://www.w3.org/TR/widgets-apis/> (Accessed 2011).

Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D. and Francalanci, C. (2011), "DashMash: A Mashup Environment for End User Development", in *Web Engineering (ICWE 2011)*

- proceedings of the 11th international conference in Paphos, Cyprus, 2011*, Springer-Verlag, Berlin, pp. 152-166.
- Dang, J. B., Hedayati, A., Hampel, K. and Taw, C. (2008), "An ontological knowledge framework for adaptive medical workflow", *Journal of Biomedical Informatics*, Vol. 41, No. 5, pp. 829-836.
- Daniel, F., Casati, F., Benatallah, B. and Shan, M. C. (2009), "Hosted Universal Composition: Models, Languages and Infrastructure in mashArt", in *Conceptual Modeling (ER 2009) proceedings of the 28th international conference in Gramado, Brazil, 2009*, Springer-Verlag, Berlin, pp. 428-443.
- Di Lorenzo, G., Hacid, H., Paik, H. Y. and Benatallah, B. (2009), "Data Integration in Mashups", *Sigmod Record*, Vol. 38, No. 1, pp. 59-66.
- Dillon, T. S., Hai, Z. G., Wu, C., Singh, J. and Chang, E. (2011), "Web-of-things framework for cyber-physical systems", *Concurrency and Computation-Practice & Experience*, Vol. 23, No. 9, pp. 905-923.
- Dillon, T. S., Talevski, A., Potdar, V. and Chang, E. (2009), "Web of Things as a Framework for Ubiquitous Intelligence and Computing", in *Ubiquitous Intelligence and Computing (UIC 2009) proceedings of the 6th international conference in Brisbane, Australia, 2009*, Springer-Verlag, Berlin, pp. 2-13.
- Ennals, R., Brewer, E., Garofalakis, M., Shadle, M. and Gandhi, P. (2007), "Intel Mash Maker: Join the web", *Sigmod Record*, Vol. 36, No. 4, pp. 27-33.
- Fosnot, C. T. (ed.) (1996), *Constructivism: Theory, Perspectives and Practice*, College Teachers Press, New York.
- Fraternali, P., Comai, S., Bozzon, A. and Carughi, G. T. (2010), "Engineering Rich Internet Applications with a Model-Driven Approach", *ACM Transactions on the Web*, Vol. 4, No. 2.
- Friedrich, M., Wolpers, M., Shen, R., Ullrich, C., Klamma, R., Renzel, D., Richert, A. and Von Der Heiden, B. (2011), "Early Experiences with Responsive Open Learning Environments", *Journal of Universal Computer Science*, Vol. 17, No. 3, pp. 451-471.
- Gaber, M. M., Zaslavsky, A. and Krishnaswamy, S. (2005), "Mining data streams: A review", *Sigmod Record*, Vol. 34, No. 2, pp. 18-26.
- Gali, A. and Indurkha, B. (2010), "The interdependencies between location expectations of web widgets", in *IADIS Multi Conference on Computer Science and Information Systems (MCCSIS 2010) proceedings of the IADIS International Conferences in Freiburg, Germany, 2010*, IADIS, pp. 89-96.
- Gasevic, D. and Devedzic, V. (2006), "Petri net ontology", *Knowledge-Based Systems*, Vol. 19, No. 4, pp. 220-234.
- Giovannella, C., Spadavecchia, C. and Camusi, A. (2010), "Educational Complexity: Centrality of Design and Monitoring of the Experience", in *HCI in Work and Learning, Life and Leisure (USAB 2010) proceedings of the 6th symposium of the workgroup Human-Computer Interaction and Usability Engineering in Klagenfurt, Austria, 2010*, Springer-Verlag, Berlin, pp. 353-372.
- Godbole, S. and Sarawagi, S. (2004), "Discriminative methods for multi-labeled classification", in *Advances in Knowledge Discovery and Data Mining (PAKDD 2004) proceedings of 8th Pacific-Asia Conference in Sydney, Australia, 2004*, Springer-Verlag, Berlin, pp. 22-30.
- Govaerts, S., Verbert, K., Dahrendorf, D., Ullrich, C., Schmidt, M., Werkle, M., Chatterjee, A., Nussbaumer, A., Renzel, D., Scheffel, M., Friedrich, M., Santos, J., Duval, E. and Law, E. (2011), "Towards Responsive Open Learning Environments: The ROLE Interoperability Framework", in *Towards Ubiquitous Learning (EC-TEL 2011) proceedings of 6th european conference of technology enhanced learning in Palermo, Italy, 2011*, Springer-Verlag, Berlin, pp. 125-138.
- Greenberg, S. (2001), "Context as a dynamic construct", *Human-Computer Interaction*, Vol. 16, No. 2-4, pp. 257-268.
- Hervas, R. and Bravo, J. (2011), "Towards the ubiquitous visualization: Adaptive user-interfaces based on the Semantic Web", *Interacting with Computers*, Vol. 23, No. 1, pp. 40-56.
- Jensen, K. and Kristensen, L. M. (2009), *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, Berlin, Heidelberg, Springer.

Jensen, K., Kristensen, L. M. and Wells, L. (2007), "Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems", *International Journal on Software Tools for Technology Transfer*, Vol. 9, No. 3-4, pp. 213-254.

Jongtaek, O. and Haas, Z. (2010), "Personal environment service based on the integration of mobile communications and wireless personal area networks", *IEEE Communications Magazine*, Vol. 48, No. 6, pp. 66-72.

Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B. and Spasojevic, M. (2002), "People, places, things: Web presence for the real world", *Mobile Networks & Applications*, Vol. 7, No. 5, pp. 365-376.

Knutov, E., De Bra, P. and Pechenizkiy, M. (2009), "AH 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques", *New Review of Hypermedia and Multimedia*, Vol. 15, No. 1, pp. 5-38.

Kopecky, J., Gomadam, K. and Vitvar, T. (2008), "hRESTS: an HTML microformat for describing RESTful Web services", in *Web Intelligence and Intelligent Agent Technology (WI-IAT 2008) proceedings of the IEEE/WIC/ACM international conference in Sydney, Australia, 2008*, IEEE Computer Society, Washington, pp. 619-625.

Krafzig, D., Banke, K. and Slama, D. (2004), *Enterprise SOA: Service-Oriented Architecture Best Practices*, New Jersey, Prentice Hall.

Laga, N., Bertin, E. and Crespi, N. (2010), "Widgets to facilitate service integration in a pervasive environment", in *Communications (ICC 2010) proceedings of the IEEE international conference in Cape Town, 2010*, IEEE, New York, pp. 1-5.

Micarelli, A. and Sciarone, F. (2004), "Anatomy and empirical evaluation of an adaptive Web-based information filtering system", *User Modeling and User-Adapted Interaction*, Vol. 14, No. 2-3, pp. 159-200.

Mödrtscher, F., Taferner, W., Soylu, A. and De Causmaecker, P. (2011), "Visualization of Networked Collaboration in Digital Ecosystems through Two-mode Network Patterns", in *Management of Emergent Digital EcoSystems (MEDES 2011) proceedings of the international conference in San Francisco, California, 2011*, ACM, New York, pp. 158-162.

Mulyar, N. A. and Van Der Aalst, W. M. P. (2005), "Patterns in colored petri nets", BPM-05-11, BPM Center.

Nelkner, T. (2009), "An Infrastructure for Intercommunication between Widgets in Personal Learning Environments", in *Best Practices for the Knowledge Society. Knowledge, Learning, Development and Technology for All (WSKS 2009) proceedings of the second world summit on the knowledge society in Chania, Crete, 2009*, Springer-Verlag, Berlin, pp. 41-48.

Ngu, A. H. H., Carlson, M. P., Sheng, Q. Z. and Paik, H. Y. (2010), "Semantic-Based Mashup of Composite Applications", *IEEE Transactions on Services Computing*, Vol. 3, No. 1, pp. 2-15.

Pohja, M. (2010), "Server push for web applications via instant messaging", *Journal of Web Engineering*, Vol. 9, No. 3, pp. 227-242.

Preuveneers, D. and Berbers, Y. (2008), "Encoding semantic awareness in resource-constrained devices", *IEEE Intelligent Systems*, Vol. 23, No. 2, pp. 26-33.

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, San Fransisco, Morgan Kaufmann.

Rozinat, A., Mans, R. S., Song, M. and Van Der Aalst, W. M. P. (2008), "Discovering colored Petri nets from event logs", *International Journal on Software Tools for Technology Transfer*, Vol. 10, No. 1, pp. 57-74.

Rozinat, A. and Van Der Aalst, W. M. P. (2006), "Decision mining in business processes", BPM-06-10, BPM Center.

Schapire, R. E. and Singer, Y. (2000), "BoosTexter: A boosting-based system for text categorization", *Machine Learning*, Vol. 39, No. 2-3, pp. 135-168.

Severance, C., Hardin, J. and Whyte, A. (2008), "The coming functionality mash-up in personal learning environments", *Interactive Learning Environments*, Vol. 16, No. 1, pp. 47-62.

Sheth, A. P., Gomadam, K. and Lathem, J. (2007), "SA-REST: Semantically interoperable and easier-to-use services and mashups", *IEEE Internet Computing*, Vol. 11, No. 6, pp. 91-94.

- Shu, N. C. (1989), "Visual programming - perspectives and approaches", *IBM Systems Journal*, Vol. 28, No. 4, pp. 525-547.
- Sire, S., Paquier, M., Vagner, A. and Bogaerts, J. (2009), "A Messaging API for Inter-Widgets Communication", in *World Wide Web (WWW 2009) proceedings of the 18th international conference in Madrid, Spain, 2009*, ACM, New York, pp. 1115-1116.
- Soylu, A., Mödritscher, F. and De Causmaecker, P. (2010a), "Utilizing Embedded Semantics for User-Driven Design of Pervasive Environments", in *Metadata and Semantic Research (MTRS 2010) proceedings of the 4th international conference in Alcalá de Henares, Spain, 2010*, Springer-Verlag, Berlin, pp. 63-77.
- Soylu, A., Mödritscher, F. and De Causmaecker, P. (2012), "Ubiquitous Web Navigation through Harvesting Embedded Semantic Data: A Mobile Scenario", *Integrated Computer-Aided Engineering*, Vol. 19, No. 1, pp. 93-109.
- Soylu, A., Preuveneers, D., Berbers, Y., De Causmaecker, P. and Desmet, P. (2011a), "Formal Modelling, Knowledge Representation and Reasoning for Design and Development of User-centric Pervasive Software: A Meta-review", *International Journal of Metadata, Semantics and Ontologies*, Vol. 6, No. 2, pp. 96-125.
- Soylu, A., Wild, F., Mödritscher, F. and De Causmaecker, P. (2010b), "Semantic Mash-Up Personal and Pervasive Learning Environments (SMupple)", in *HCI in Work and Learning, Life and Leisure (USAB 2010) proceedings of the 6th symposium of the workgroup Human-Computer Interaction and Usability in Klagenfurt, Austria, 2010*, Springer-Verlag, Berlin, pp. 501-504.
- Soylu, A., Wild, F., Mödritscher, F., Desmet, P., Verlinde, S. and De Causmaecker, P. (2011b), "Mashups and Widget Orchestration", in *The International Conference on Management of Emergent Digital EcoSystems (MEDES 2011) in San Francisco, California, 2011*, ACM, pp. 226-234.
- Spiekermann, S. (2008), *User Control in Ubiquitous Computing: Design Alternatives and User Acceptance*, Aachen, Shaker Verlag.
- Srblic, S., Skvorc, D. and Skrobo, D. (2009), "Widget-Oriented Consumer Programming", *Automatika*, Vol. 50, No. 3-4, pp. 252-264.
- Taivalsaari, A. (2009), "Mashware: The future of web applications", Sun Microsystems.
- Tan, W., Fan, Y. S., Zhou, M. and Tian, Z. (2010), "Data-Driven Service Composition in Enterprise SOA Solutions: A Petri Net Approach", *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 3, pp. 686-694.
- Tsoumakas, G. and Katakis, I. (2007), "Multi-label classification: An overview", *International Journal of Data Warehousing and Mining*, Vol. 3, No. 3, pp. 1-13.
- Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J. and Vlahavas, I. (2011), "MULAN: A Java Library for Multi-Label Learning", *Journal of Machine Learning Research*, Vol. 12, pp. 2411-2414.
- Tuchinda, R., Knoblock, C. A. and Szekely, P. (2011), "Building Mashups by Demonstration", *ACM Transactions on the Web*, Vol. 5, No. 3.
- Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R. and Decker, S. (2010), "Sig.ma: Live views on the Web of Data", *Journal of Web Semantics*, Vol. 8, No. 4, pp. 355-364.
- Valjataga, T. and Laanpere, M. (2010), "Learner control and personal learning environment: a challenge for instructional design", *Interactive Learning Environments*, Vol. 18, No. 3, pp. 277-291.
- Van Der Aalst, W. M. P. (2011a), "Do Petri Nets Provide the Right Representational Bias for Process Mining?", in *Applications of Region Theory (ART 2011) proceedings of the workshop in Newcastle, UK, 2011*, CEUR-WS.org, pp. 85-94.
- Van Der Aalst, W. M. P. (2011b), *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Berlin Heidelberg, Springer.
- Van Der Aalst, W. M. P., Pesic, M. and Song, M. (2010), "Beyond Process Mining: From the Past to Present and Future", in *Advanced Information Systems Engineering (CAiSE 2010) proceedings of the 22nd international conference in Hammamet, Tunisia, 2010*, Springer-Verlag Berlin, Berlin, pp. 38-52.
- Van Der Aalst, W. M. P., Van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G. and Weijters, A. (2003), "Workflow mining: A survey of issues and approaches", *Data & Knowledge Engineering*, Vol. 47, No. 2, pp. 237-267.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

Van Der Aalst, W. M. P., Weijters, T. and Maruster, L. (2004), "Workflow mining: Discovering process models from event logs", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 9, pp. 1128-1142.

Van Harmelen, M. (2008), "Design trajectories: four experiments in PLE implementation", *Interactive Learning Environments*, Vol. 16, No. 1, pp. 35-46.

Vidal, J. C., Lama, M. and Bugarin, A. (2010), "OPENET: Ontology-based engine for high-level Petri nets", *Expert Systems with Applications*, Vol. 37, No. 9, pp. 6493-6509.

Virzi, R. A. (1992), "Refining the test phase of usability evaluation - how many subjects is enough?", *Human Factors*, Vol. 34, No. 4, pp. 457-468.

Widmer, G. and Kubat, M. (1996), "Learning in the presence of concept drift and hidden contexts", *Machine Learning*, Vol. 23, No. 1, pp. 69-101.

Wild, F., Mödritscher, F. and Sigurdarson, S. E. (2008), "Designing for Change: Mash-Up Personal Learning Environments", *eLearning Papers*, Vol. 9.

Wilson, S., Daniel, F., Jügel, U. and Soj, S. (2011), "Orchestrated User Interface Mashups Using W3C Widgets", in *Web Engineering (ICWE 2011) proceedings of the 11th international conference in Paphos, Cyprus, 2011*, Springer-Verlag, Berlin, pp. 49-61.

Wu, X. T. and Krishnaswamy, V. (2010), "Widgetizing Communication Services", in *Communications (ICC 2010) proceedings of the IEEE International Conference on in Cape Town, South Africa, 2010*, IEEE, New York, pp. 1-5.

Yang, S. C. (2001), "Synergy of constructivism and hypermedia from three constructivist perspectives - Social, semiotic, and cognitive", *Journal of Educational Computing Research*, Vol. 24, No. 4, pp. 321-361.

Yu, J., Benatallah, B., Casati, F. and Daniel, F. (2008), "Understanding mashup development", *IEEE Internet Computing*, Vol. 12, No. 5, pp. 44-52.

Zhiqing, X., Si, W., Heqi, Y., Zhenyu, W., Hao, C., Chunhong, Z. and Yang, J. (2010), "A new architecture of web applications-The Widget/Server architecture", in *Network Infrastructure and Digital Content (IC-NIDC 2010) proceedings of the 2nd IEEE international conference in Beijing, China, 2010*, IEEE, New York, pp. 866-869.