Toward Process Mashups: Key Ingredients and Open Research Challenges

Florian Daniel
University of Trento
Via Sommarive 14
38123 Povo (TN), Italy
daniel@disi.unitn.it

Agnes Koschmider
University of Pretoria
Department of Computer Science
0002 Pretoria
akoschmider@cs.up.ac.za

Tobias Nestler
SAP Research Center
Dresden, Germany
tobias.nestler@sap.com

Marcus Roy
SAP Research Australia
168 Walker Street
2060 North Sydney, Australia
m.roy@sap.com

Abdallah Namoun
Centre for Service Research
University of Manchester
Manchester, M13 9SS, UK
abdallah.namoune@mbs.ac.uk

ABSTRACT

Over the last few years, the mashup community has grown significantly, and mashup development has matured substantially compared to the initial hacking practices. Mashups as applications have specialized into *data mashups*, *service mashups*, or *user interface mashups* – although these terms lack a common agreement on definitions – while other types of mashups can still be identified. In fact, recently the term *process mashup* emerged, yet, again, its meaning is everything but clear.

Intrigued by this latter idea, in this paper we try to understand what process mashups are. We identify *three dimensions* that distinguish process mashups from most of the current types of mashups and we show that exploring them leads to *a set of new types of mashups*, which are the actual basis for the development of process mashups. For each of these new types of mashups, we provide a discussion, discuss suitable application scenarios and show tool support, so as to highlight *challenges and open issues*.

1. INTRODUCTION

During the last few years, Mashups have attracted considerable attention as situational applications composing existing and reusable Web resources to solve a new and immediate problem. In this context, the development of mashup applications is considered relatively easy and flexible compared to traditional application composition by primarily addressing casual and less experienced developers. Mashups tools leverage the potential of the long tail of users, folksonomic features and Web 2.0 techniques, leading to a large amount of mashup applications being developed, e.g., see ProgrammableWeb[1]. Although such mashup development can be considered quite popular in the community, it seemed to fall short of achieving significant impact in an enterprise context. To underline, Gartner [4] considered Enterprise Mashups as one of the top ten strategies in 2008 and 2009 but not for 2010. This raises the question of possible factors that could have caused the lack of success for Enterprise Mashups.

From an enterprise Service-Oriented Architecture (eSOA) point of view, organizations offer Web Services to expose business-relevant data and functionality from existing legacy applications. Therefore, business process automation based on activities implementing these Web Services is an essential requirement for enterprises to run crucial business operations. Currently, the application development mostly addresses skilled developers and is still a costly and timely effort vulnerable to frequent changes.

However, business processes are typically characterized by the involvement of multiple user roles and a workflow-style composition. As a simple example, a leave request application might require multiple views, e.g. for an employee to enter the request as well as for a manager to review and approve or reject the request. Based on the manager's decision, an individual notification might be sent to the employee. As a minimum, this scenario requires support for multiple users and a central workflow to guide the manager through a sequence of steps or pages that require his/her attention.

We argue that existing mashup development approaches do not adequately address the creation of applications and capabilities as described in the above scenario. Within this paper we refer to these kind of resulting applications as *Process Mashups* – a term first introduced by [17]. Moreover, its definition is still fairly vague, posing the question: what actually defines a process mashup? To better understand the definition and future requirements of process mashups, we introduce three dimensions, i.e., *multi-user* support, *multi-page* navigation, and *workflow* support, which we consider significant characteristics to describe this class of mashups. In this context, each combination of our dimensions represents a specific class of mashups, e.g. ranging from single-user, single-page and no workflow to multi-user, multi-page and workflow-supported mashups, where we understand only the latter as process mashup. Based on our dimensions, we investigate existing mashup development tools and classify them into the relevant mashup class. Starting from this classification, we

[1] `http://www.programmableweb.com/` (5198 Mashups; Sept'10)

outline future research challenges that we have to solve in order to turn process mashups into practice.

The remainder of this paper is as follows. In Section 2, we provide a background on tools and technologies supporting the management and execution of processes as well as first approaches towards Process Mashups. Section 3 introduces our dimensions followed by the classification of existing mashup tools in Section 4. In Section 5, we discuss the research challenges before we conclude the paper in Section 6.

2. BACKGROUND

Commonly we define a *mashup* as a web application that integrates data, application logic, and pieces of user interfaces (UIs) [7]. To understand what kind of new perspectives a process mashup should reveal we have to understand characteristics of a process and a workflow and how they can stimulate a mashup.

A business process is a set of activities that can be represented through visual languages. A workflow is an executable part of a process that consists of several activities and defines a series of tasks that need to be managed by different resources (individuals or application components) [14]. A workflow assigns tasks to users and guides them through these tasks.

Business processes mainly focus on the control-flow perspective and the technical level of a Service-to-Service interaction. The extension BPEL4People [2] attempts to better support human interaction by introducing the concept of the people task as first-class citizen into the orchestration of services. The extension is tightly coupled with the WS-HumanTask [1] specification, which focuses on the definition of human tasks, including their properties, behavior and operations used to manipulate them. BPEL4People supports people activities in form of inline tasks (defined in BPEL4People) or standalone human tasks accessible as web services. In order to control the life cycle of service-enabled human tasks in an interoperable manner, WS-HumanTask also comes with a suitable coordination protocol for human tasks, which is supported by BPEL4People. The two specifications focus on the coordination logic only and do not support the design of the UIs for task execution.

Service composition is still a complex, time consuming, and error prone process requiring strong modeling abilities and a deep knowledge in terms of existing standards (WS* protocol) and composition languages. The need for situational applications (i.e. applications that come together for solving some immediate business problems) to address individual and heterogeneous needs as well as the shift to more flexible and dynamic business environments encourage the idea of lightweight composition and mashups in particular [9].

While most mashups are still hand-coded, graphical mashup platforms aiming for a simplified web development to support less experiences developers. The graphical composition style of platforms like Yahoo! Pipes [2], Open mashup [3] or Marmite [15] supports the ad-hoc creation of simple web applications by combining data from heterogeneous resources like web feeds, web pages and web services. Each resource is represented as a component or widget and can be dragged and linked with others. Thereby, the output of one component serves as the input of another (Piping and Wiring). The majority of platforms (overview provided by [8]) focuses on the development of data-oriented mashups in the form of value-added services, feeds or single page web applications (no views are supported). Control-flow is typically not considered and only single users are addressed.

Several tools claim to be a mashup platform and to support the modeling and execution of business processes as one feature. Tools like the SOA4All Studio [10] or Serena Mashup Composer[4] provide process modeling editors among other abstraction layers and views on an application. LiquidApps[5] supports the integration of BPEL process descriptions as a foundation for the generation of screens and forms. Even though all of these tools manage business processes they require a deep understanding of development concepts and can be considered as professional development environments.

The concept Process Mashup has already been discussed in the literature. [17] introduces the concept of *process mashups* as the next type of mashups that consider besides the data and presentation layer also the integration of business processes. Following this idea, [16] proposes process-oriented mashups that support the coordination or automation of tasks and activities by modeling the control flow. The authors discuss motivating examples for this new category of situational applications, point out interesting challenges like the provision of reusable pattern and artifacts but lack concrete concepts or solutions about how to achieve this objective. Another approach that promises to combine data and process flow is Mashlight [3] where predefined widgets (Mashlight Blocks) can be ordered along a process flow.

From our understanding, a Process Mashup is more than adding business processes to the data and presentation layer. The analysis reveals a process mashup beyond the current proposals. A process mashup should tackle multiple users and organize human tasks. If work is properly structured into tasks, this implies the need for multiple pages or views on a process even in presence of only one user. Thus, a process mashup should also address multiple pages. The junction of all together allows a controlled interwoven composition of the three basic ingredients data, functionality and UI that is currently not at all supported.

3. PROCESS MASHUPS: INGREDIENTS

Although mashups are typically still simple, one-page applications, compared with traditional data and application integration practices mashups feature one significant new innovation: they foster integration – and, hence, *reuse* – also at the level of UIs, i.e., the presentation layer. Of course, this practice is still in an early stage of maturity, and suitable support in terms of readily available components or APIs is still lacking in most of the cases. The best example of API that comes with its own UI is Google Maps with its relatively easy to use JavaScript programming interface. Yet, in many cases UI elements are still scrapped or extracted from web pages, typically without the actual provider of the web page knowing about it.

The discussion in the last section shows that business processes have some typical characteristics that also process mashups will have to support. Specifically, we can identify *three new dimensions*, i.e., support for multiple users, support for navigation among multiple connected

[2] `http://pipes.yahoo.com`
[3] `http://www.open-mashups.org/`
[4] `http://www.serena.com/products/mashup-composer/`
[5] `http://www.liquidappsworld.com/index.php`

pages, and support for workflows. To be able to analyse existing mashup approaches in regard to this new dimensions we discuss each of them in the following.

Note that in this paper we explicitly distinguish between *workflow* and *control flow* in order to distinguish the coordination of only human actors and their work tasks (in the former case) from the generic coordination of (computing) tasks (in the latter case).

3.1 Multiple users

Supporting multiple users means allowing them to *concurrently* operate a same instance of a mashup application. This is different from sharing a mashup (e.g., a Yahoo! pipe) among multiple users, which in practice just means allowing them to run their very own instance of the mashup.

Identifying and authorizing users and assigning them individual work items (tasks) is one of the drivers that led to the emergence of workflow management and business process management systems. Activities in a process model can commonly by associated with actors, e.g., via swim lanes as in BPMN or via suitable parameters. This association is usually performed via roles, which allows the late binding of actual actors to a process instance (e.g., at deployment time or at runtime).

Such kind of collaborative, multi-user mashups are currently not supported. Current mashups do not allow multi-users to concurrently or cooperatively work together (e.g., via different UI views of the mashup) nor do they offer role-based access mechanisms to the mashup. Instead, so far they mostly focus on single (isolated) users where each user has his/her own private instance of the mashup.

Supporting multi-user mashups will require addressing the following challenges (among others):

- **Concurrent access**: How can we support concurrent access of multiple users to a same view of a mashup application? Concurrent in this case means that all users work together on a same view of the mashup. For instance, we can imagine a mashup in which the employee co-browses his options for the leave request together with his manager and they jointly decide when it is fine for both.
- **Role-based access**: How can we support cooperative access of multiple users to a mashup application via different views on the application? The goal here is to allow multiple users to cooperate, but by confronting each user with mashup information that is relevant for his/her role and activities only. As an example, the employee and the manager may agree on the leave request in such a way that the employee selects his configuration alone, and the manager then has another view just to accept or reject the proposal.

3.2 Multiple pages

Implementing mashups that are composed of multiple pages means being able to organize the integrated components of the mashup into a possibly *hierarchical navigation structure* that can be explored by the users via hyperlinks.

Business process or workflow management systems usually provide their users with user interfaces (e.g., input forms) that are tailored to the work item to be done. If work is properly structured into tasks, this implies the need for multiple pages or views on a process even in presence of only one user. If multiple users cooperate via a process, this requirement becomes even more stringent.

Today, most of the mashups that one can find (e.g., on `programmableweb.com`) are web applications with a single page. Multiple pages are slightly used for Enterprise Mashups where multiple pages are generated by using dialog controls such as tabs. Besides the fact that in such a case mashup composers should hold web design criteria for each page, the tab paradigm is not more than an acceptable compromise between ease of implementation and provided features. Also, the problem of current multi-page mashups is that they are created in an ad-hoc manner, resulting into very different approaches to distribute content over pages.

A good multi-page mashup answers the following challenges:

- **Navigation structure**: How do we structure integrated data, APIs, and UIs into a structured set of pages? Connected to this, it is also necessary to understand how it is possible to define a good navigation structure on top of these pages, i.e., how to interlink pages. For instance, the employee could be provided with two pages, one dedicated to the description of the leave request and one to the appointment of a substitute; this may enhance the usability of the mashup.
- **Navigation state**: How do we keep the state of the navigation in a multi-page mashup? Differently from traditional web applications, in which the URL encodes the navigation progress through the application, mashups may integrate a variety of JavaScript or AJAX components (e.g., a Google Map), which pose new requirements if we want to keep track of the user's selections, since they typically operate outside the control of the browser.

3.3 Workflows

Providing workflow support in a mashup means being able to specify a *control and data flow over human tasks*. That is, it requires being able to define sequences, branches, or conditional executions of work items.

In the early 90ies, establishing such kind of process logic as own modeling concern and detaching its execution from monolithic software systems was one of the major contributions of workflow management systems. Typically, modern process modeling formalisms distinguish between control flow (i.e., the order of tasks) and data flow (i.e., the way data is propagated among tasks). The coordination of multiple actors may happen via control flow only, since business data may also "flow" only in the real world from one actor to another, yet the process needs to progress.

Today's data mashups (like Yahoo! Pipes Mashups) or service compositions are already highly process-based without however implementing a control flow. In fact, these types of data mashups are typically data flow based only. Very likely the reason for this choice is simplicity and the lack of the need to coordinate multiple actors. UI-based mashups are more event-based, yet when it comes to the integration of web services, control flows to express the service orchestration logic is typically needed.

Mashing up people, UIs, data, and services and mapping them to an overall workflow requires again addressing some peculiar challenges:

- **Workflow**: How do we define a workflow for human actors over mashups or components of mashups? Answering this question implies deciding the granularity at which we want to assign actors and

control the state of the progress (e.g., page level vs. component level) and deciding whether to use process data/variables to keep track of the process' state or not. In our example, we can, for instance, decide to put the employee's request into a cycle with the manager's decision, until an agreement is reached.

- **Data flow**: How do we propagate data from one task to another task (if any)? Coordinating tasks may imply forwarding business data (e.g., an electronic form) from one task to another. This flow of business data between different mashup tasks and users, resulting from the need to circulate data (e.g., exchanging, manipulating and evolving), must be efficiently coordinated. The data flow between the employee and the manager can, for example, be boiled down to the proposed leave period message and the acceptance/rejection message.
- **Integration of data, services, UIs, and people**: If things get more complex, how do we coordinate not only human actors but also data, web services, and the UIs the human actors need to interact with? Doing so requires conciliating the needs of services (orchestration) with those of UIs (synchronization) and those of human actors (workflows).

3.4 The New Perspective

In order to better understand how process mashups could now look like, we need to put the three previous ingredients or dimensions together. The output of this junction is illustrated in Figure 1. The illustration shows an interesting result, which we think deserves further investigation before being able to actually talk about process mashups: there are a *variety of different mashup types* with different characteristics that can be seen as intermediate steps (also in terms of the mashup platforms that can help implementing them) to be investigated, before approaching the full process mashup problem.

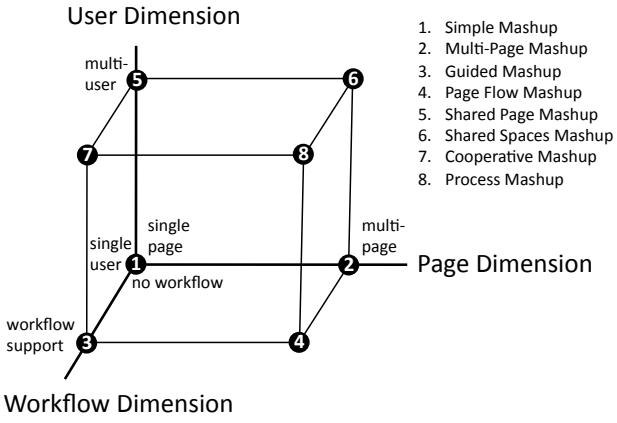

Figure 1: Types of mashups

4. CLASSES OF MASHUP TOOLS

We define and discuss each of the above types in the following section, keeping a special eye on how these types of mashups are supported with existing mashup tools. Each junction in the illustration may require different development approaches and solutions. Although, we are not aware of any tool that implements a process mashup, each of these mashup types is fully justified since they provide support for different application scenarios.

4.1 Simple Mashups

Definition: This mashup type exclusively addresses Single User / Single Page / No Workflow
Example: A simple mashup could, for instance, allow the employee to have an integrated picture of his/her colleagues' leave requests, so as to tailor their own request to the company's needs. Data could be sourced from a company-internal repository via a dedicated web service, and results and inputs can be rendered in the web page via suitable UI components. There is no direct interaction or cooperation with the manager, and the mashup rather serves an informational purpose.
Representative platform: The mashArt platform [5], for example, focuses on so-called *hosted universal composition* for the web. The platform comes with models, languages, and a composition paradigm that allows one to abstract from low-level implementation details and to compose components that are characterized by heterogeneous technologies, ranging from simple feeds to complex web services and UI components, within the same development environment. In doing so the platform conciliates the need for orchestration of process-oriented service composition with the need for synchronization of event-based UI development via a unique event- and data flow-based composition logic. The platform also supports the definition of complex, e.g., asynchronous, service orchestration logics. Mashups can easily be dragged and dropped together, previewed, stored, and executed in a hosted fashion.

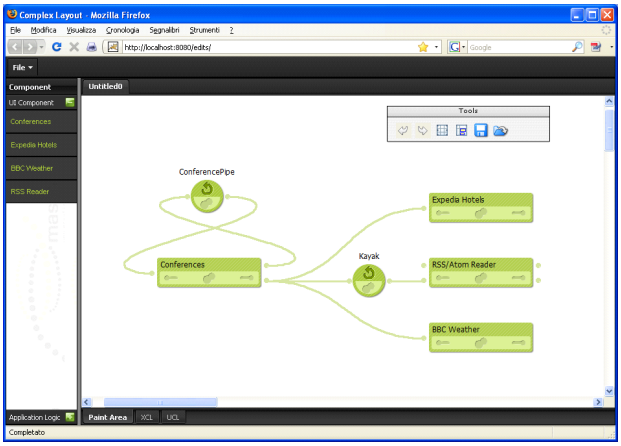

Figure 2: The mashArt editor at work

Figure 2 shows the mashArt editor at work. Data/web services (the round constructs) can easily be integrated with user interface components (the rectangles) by piping and wiring events of one component to operations of other components. UI components can be associated with placeholders inside common HTML templates, so that they can be rendered to the user.
Other platforms: This is the most studied class of mashups as of today. There are a variety of different tools, among we mention Yahoo! Pipes[6] for data mashups, and Intel Mash Maker[7] for UI mashups.

4.2 Multi Page Mashups

Definition: This mashup type exclusively implements Single User / Multi Page / No Workflow

[6] `http://pipes.yahoo.com/pipes/`
[7] `http://mashmaker.intel.com`

Example: A multi-page mashup in the context of our leave request example can provide an overview of incoming leave requests in form of a list. After selecting one specific entry within the list the mashup switches automatically to another page and displays additional details about the requesting employee. Users of such mashups are working without any interaction or cooperation with other roles and are not guided in his/her work.
Representative platform: The EzWeb platform allows users to create Mashups by combining or *wiring* gadgets. In this context, each gadget can be understood as a "mini application" (e.g. Amazon Shopping Cart) developed using the FAST visual storyboard application [11]. Furthermore, gadgets can consist of multiple screens that are arranged according to a screenflow (see Fig. 3). The screenflow is modeled in a storyboard-like way by placing predefined screens (e.g. Amazon Product List) into the working area. However, connections between screens are not explicitly modeled but automatically generated based on a mapping of their input and output types and semantics. This means the workflow in FAST does not have a control flow per se but is implicitly defined by the data flow mapping. Although FAST does not support multi-user roles, it allows a single user to navigate seemingly free through screens/pages.

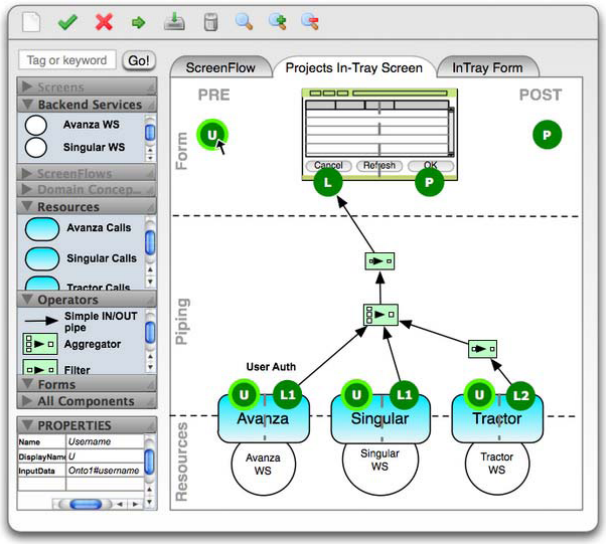

Figure 3: Gadget development with FAST [11]

4.3 Guided Mashups

Definition: This mashup type exclusively implements Single User / Single Page / Control-Flow
Example: This mashup class offers user guidance in order to fulfill his/her task. Such a mashup reacts on user's inputs and gradually provides the next possible activities. Imagine the employee wants to request a leave. He/she invokes the leave request. Instead of showing, for instance, a PDF with interactive formular fields, this mashup type guides the user through this process. After selecting his/her name, the mashup shows the employee's leave entitlement for the year. Subsequently, the employee has to decide what kind of leave he/she requests (holidays, sabbatical leave). In case of a sabbatical leave a text box is opened that asks the employee to state a reason. Next, the employee selects his/her period for the leave. A calendar is opened and highlights days that are free for a leave in green color. The remaining days are marked in red. Finally, the employee must electronically sign the request form. If all fields were filled correctly, a send button is shown.
The benefit of this mashup type is that users are guided through their work.
Representative platform: We are not aware of any tool for guided mashups. Guided applications for leave requests might also be implemented with standard software. In section 2 we discussed the benefits of situational applications compared to other application types.

4.4 Page Flow Mashups

Definition: This mashup type addresses Single User / Multi Page / Control-Flow
Example: Following our running example a mashup of this category could support the manager in handling leave requests. One page of the application can provide a list of incoming leave requests. After selecting one specific entry the option to approve or decline the request appears. To support the decision process, the manager can switch to another page of the application to request additional details about the employee as well as an overview of all the leave requests applied so far. A third page provides statistics about all leave requests of the employers' department for a specific period of time. The manager is solely working on the pages without any interaction with the employee.
Representative platform: The ServFace Builder[12] is a web-based authoring developed in the frame of the EU-funded project ServFace[8]. The tool supports non-programmers in the design and creation of service-based interactive applications in a WYSIWYG manner. It applies the approach of service composition at the presentation layer, in which applications are built by composing web services based on their frontends, rather than application logic or data. Applications are designed as a set of pages that can be connected to create a navigation flow. The tool supports different navigation styles like wizards (Guided Procedure) or a free navigation like the one known from traditional websites. Services are represented as form-based UIs and can be connected across pages in order to define a data flow.

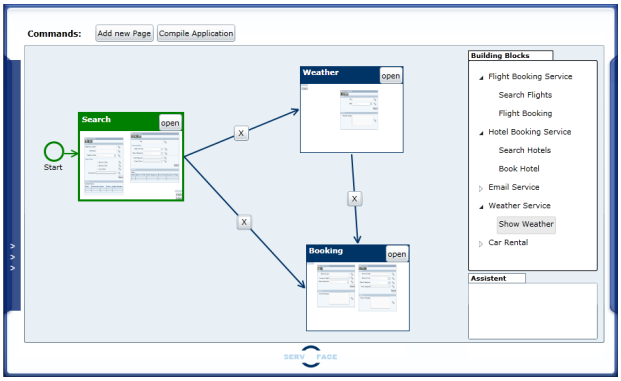

Figure 4: Page Flows within the ServFace Builder

Figure 4 shows the ServFace Builder at work. The application has three pages, whereby each page represents one dialog visible on the screen. Connections between pages build the control flow of the application and create an entry in a navigation menu, which is part of each page. The

[8] `http://www.servface.eu`

tool supports the user in the connection of pages according to the chosen navigation style.

4.5 Shared Page Mashups

Definition: This mashup type addresses Multi User / Single Page / No Workflow
Example: On `Programmableweb.com` one can find one mashup addressing multi-users. The Amazon.com Shopping Together mashup allows to shop together through the MSN Messenger platform. While browsing the online catalogue users can chat together.
A shared page mashup, in the context of the leave request scenario, would allow employees to simultaneous work on a single request. The employees would not be guided in their work.
Representative platform: We are not aware of any mashup tool for this category.

4.6 Shared Space Mashups

Definition: This mashup implements Multi User / Multi Page / No Workflow
Example: Such a mashup allows multiple users to concurrently share a space.
Let us consider the following scenario. An employee can edit on the first page his/her leave request, on a second page he/she can preview previous leave requests. The employee can share his/her space with the manager and, if applicable, with his/her temporary replacement. The manager can even preview the previous leave requests of the employee, which is prohibited for the temporary replacement. In case that the employee has selected days that are overlapping with leaves of other employees an additional page is opened in the space of the manager showing the conflicts. There is no guidance for their work.
Representative platform: The IBM Mashup Center[9] is a collection of tools allowing users to create (Enterprise) Mashups. Main customers of this tool are businesses looking for a quick access to a consolidated view of information. The recent version of the IBM mashup Center (V2.0) comes with collaboration features called *spaces* allowing users to create and (collaboratively) share mashup pages. Spaces are collections of mashup pages assembled together and providing a navigation between the pages. The spaces concept supports concurrent access to the mashup without any user guidance. Additionally, the administrator of the mashup can establish a role-based access.

4.7 Cooperative Mashups

Definition: This mashup type exclusively addresses Multi-User / Single Page / Workflow
Example: A mashup of this category would allow one employee to resolve conflicts about a leave request in case another employee wants to travel at the same time. They jointly view the conflict and can decide which leave request should be deleted. The interaction between the employees is guided.
Representative platform: Gravity[10] is a lightweight, collaborative and web-based business process modeling client targeting non-BPM-experts to create immediate applications based on business process modeling. It allows multiple users to simultaneously model the underlying business process (model view) and the corresponding user interfaces (application-design view) applying mashup-style composition techniques, e.g., drag & drop and drawing lines (see Figure 5). Also, changes in the application-design view are instantaneously propagated to the model view and vice versa. Hence, Gravity supports multi-user involvement and defines a control flow based on the underlying business process model as well as a data flow based on the mapping of fields in the application design view resulting into an executable application. Although Gravity does not literally support multi-pages, it gets quite close to our definition of a process mashup.

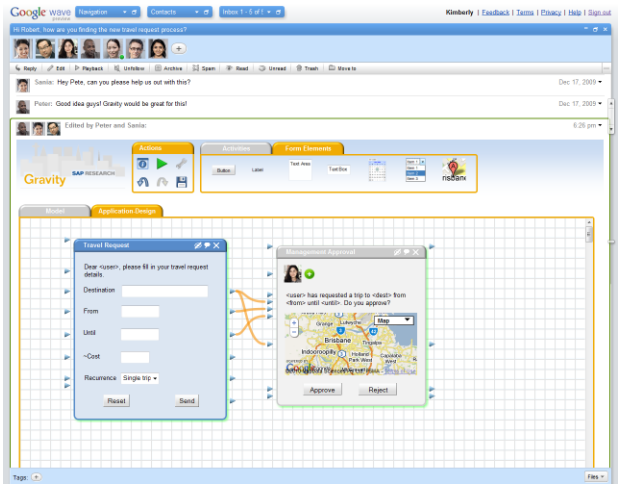

Figure 5: Gravity within the Google Wave environment

4.8 Process Mashups

Definition: Multi-User / Multi-Page / Workflow
Example: Finally, process mashups are the most complex of the combinations we identify. We can, for instance, have a mashup that provides the employee with a mashed up web page that allows him/her to configure his preferred flight and hotel combination and the manager with a page that allows him/her to inspect the employee's choice and to approve or reject it. The mashup may further put the two pages/tasks into a cycle that terminates only upon the approval of one of the choices.
Representative platform: It is hard to find a good representative for this type of mashup. Maybe the MarcoFlow platform [6], which is based on the idea of distributed orchestration of UIs, is a first attempt into that direction. The platform features an application development approach that allows one to bring together UIs, web services, and people in a single orchestration logic, language, and tool. MarcoFlow covers three main phases of the software development life cycle: design (through a dedicated, visual editor), deployment (through a set of code generators), and execution (through a distributed runtime environment). The key idea to approach the coordination of (i) UI components inside web pages, (ii) web services providing data or application logic, and (iii) individual pages (as well as the people interacting with them) is to split the coordination problem into two layers, i.e., intra-page UI synchronization and distributed UI synchronization and web service orchestration, and to provide runtime environments to both (a client-side JavaScript environment plus a BPEL engine in the backend).

Development of distributed UI orchestrations is supported via an extended Eclipse BPEL editor, as illustrated in Figure 6. The editor supports the specification of com-

[9] `www.ibm.com/software/info/mashup-center/`
[10] `http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/17826`

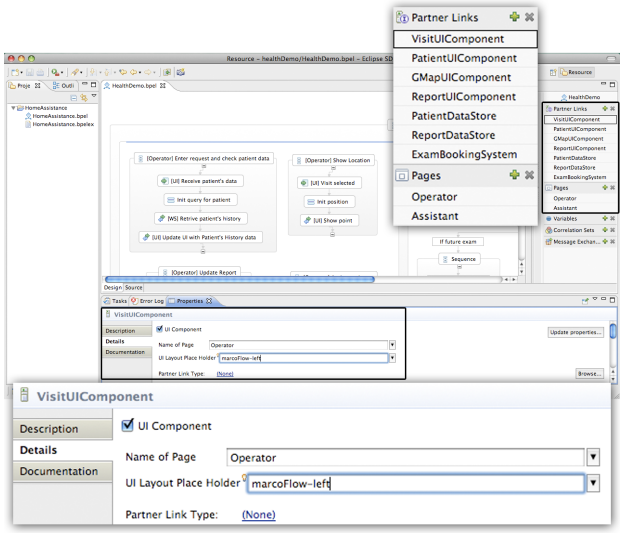

Figure 6: The extended MarcoFlow Eclipse BPEL editor; UI-specific extensions are highlighted.

munications among services and UI components (both abstracted in terms of WSDL interfaces), the collection of UI components into pages, and the assignment of actors to pages.

Other platforms: Even though it does not really support the composition of UIs, the use of JOpera, as exemplified by Pautasso [13], could allow the setup of the interaction logic among components and multiple users. UIs are developed as traditional web applications and accessed via RESTful service interfaces.

5. DISCUSSION

The above casuistry shows that while in all the eight cases of mashup applications it is always possible to provide meaningful examples of mashups, it is not always possible to also identify a mashup tool that supports this specific type of mashup. Specifically, to the best of our knowledge we have not been able to identify any publicly available mashup tool for the development of *guided mashups* and *shared page mashups*.

Yet, one of the key challenges in supporting the development of mashups is to provide appropriate tool support. Current trends and research show that simplified visual design metaphors and end-user friendliness are important criteria for a good mashup platform. In fact, one of the key drivers for mashups is their simplified and rapid development, especially of situational applications, which cannot be done with traditional composition approaches due to their complexity in terms of standards and composition languages. Simple development metaphors and end-user orientation need to be guaranteed even if the resulting mashups incorporate more features and complexity, such as it is the case of *process mashups*, otherwise these tools become more and more powerful development instruments for experienced developers and loose their benefit to non-programmers. If mashup platforms increase their complexity too much, this advantage will disappear, and maybe they are no longer "mashup platforms".

In order to better understand this concern, we first of all need to define what we mean with the term *mashup platform*:

> A *mashup platform* is a composition and runtime environment for applications (mashups) that may span all *three layers of the software stack* (data, application logic, and UI) and that typically *does not require programming skills* (e.g., by keeping technology as simple as possible and adopting visual development metaphors).

Given this definition, the border between what should be considered a mashup platform and what not should be less blurred: Ease of use and end-user focus, i.e., no need for programming skills, are addressed by tools like FAST, mashArt, Gravity, and ServFace Builder. More precisely, FAST and mashArt support the less experienced developer in the development of complex applications. They still require some technical knowledge, but they clearly focus on a rapid and lightweight resource composition. Gravity and ServFace Builder require less skills and support the development of applications by end-users and non-programmers. However, this simplicity does not come for free, and their complexity or expressiveness is not as powerful as some of the other platforms. The most complex tool in our analysis is MarcoFlow. The extended Eclipse BPEL editor provides all the required functionality to realize all the three dimensions of a process mashup, but clearly looses the lightweight composition perspective in that it uses an extended version of BPEL, a language that is already hard to learn by skilled developers. So, according to our above definition, the MarcoFlow system cannot really be considered a mashup platform.

Besides effective mashup editors for process mashups, it is further important to reason about what kind of *runtime support* such kind of mashup applications will require. From the discussion of the different types of mashups seems to emerge one important aspect that may change the way mashups are perceived: moving from mashups that do not support any kind of human workflow to mashups that allow the coordination of and cooperation among multiple actors means that process mashups loose the typical situational nature of today's mashups.

First of all, coordinating different actors and their work items typically requires software support that is long-running and, in some cases, stateful. This, in turn, raises the need for some kind of execution environment for process mashups that is able to keep alive instances of process mashups and to progress their state upon user interactions. If the required process logic is only short-living, a simple application may serve the purpose. But if things get more complex, an engine that is able to handle long-running process logic and multiple concurrent instances will be needed.

Next, dealing with multiple users in the execution environment raises the need for adequate identification and authentication mechanisms, a requirement that definitely adds complexity to both the execution environment and the mashups running in it. Roles may be needed to organize users into groups, access rights to restrict access of individual users or groups to features of the execution environment or to the mashups themselves. To recall, mashup development has become popular due to its simplicity by purposely avoiding any security and reliability requirements among others. Although, more powerful mashup platforms and execution environments have the benefit to create more expressive, long-running and possibly stateful mashup applications involving various user roles/groups, the required set of skills and knowledge to use them effectively is counter-productive to the mashup idea.

RESTful or lightweight approaches to these problems could help simplify some of these issues in terms of technology, but the conceptual complexity of full-fledged process mashups cannot be further reduced. In fact, even with the simplest development instrument a mashup designer still needs to know how to structure human tasks into pages that can be assigned to individual actors, how to coordinate the work of multiple actors, which data the actors need to perform their tasks, how to propagate such data from one actor to another, etc. That is, developing process mashups is a non-trivial effort consolidating various stakeholder tasks such as application planning (what user roles, what functionality must be supported), design (UI composition), composition (workflow, coordination of actors), implementation (programming) and execution (runtime configuration). Hence, mashup developers performing these task must be quite versatile, having a bit of all including programming knowledge, otherwise it will be hard to implement anything that works.

6. CONCLUSION

Current mashups are in most cases single page applications for one user. They represent the simplest form of mashups that require the least amount of user skills to create them. Resulting mashups indeed solve situational and less complex problem with a high degree of reusability. However, the need for mashup applications to solve more specific and complex problems led to mashup tools supporting multi-users and/or incorporating multi-pages, which are only applicable in some scenarios, but not all. Specifically in an enterprise context, the trend to integrate process functionality within mashups motivated us to introduce three new dimensions that helped us to understand the concept behind process mashup. Therefore, we derived the new dimensions from characteristics of processes and workflows and discussed how mashups could benefit from them. Based on the investigation of existing mashup platforms, we started a classification according to our dimensions along which we derived research challenges that have to be met for future process mashup development.

In general, we argue that process mashup development is a challenging task that requires advanced skills and extended software support, which quickly goes beyond the original idea of mashup development. Although effort can (and needs to) be invested in suitable HCI research in order to come up with intuitive mashup paradigms and metaphors, which will ease the interaction with mashup tools, this won't be able to simplify much further a task that is intrinsically complex. In fact, we stressed the additional requirements and constraints needed to realize process mashups, which position process mashups at the boundary between mashups and traditional application development. Hence, future mashup platforms aiming at the development of process mashups will have to consider a trade-off between the expressiveness and capabilities of mashups and the required skill set of users needed to effectively utilize the mashup platform to create and run them. In short, we believe that process mashups will not replace core business process management systems, but their increasing capabilities definitely promise new opportunities for a widespread adoption of mashups within enterprise environments.

Acknowledgements

This work is supported by the European research projects ServFace, Omelette and SOA4All.

7. REFERENCES

[1] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP. Web Services Human Task (WS-HumanTask), Version 1.0. Technical report, June 2007.

[2] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP. WS-BPEL Extension for People (BPEL4People), Version 1.0. Technical report, June 2007.

[3] M. Albinola, L. Baresi, M. Carcano, and S. Guinea. Mashlight: a Lightweight Mashup Framework for Everyone. In *Proceedings of WWW*, 2009.

[4] D. Cearley and C. Claunch. Top 10 strategic technologies for 2010. `www.gartnerinfo.com/g17.cgi/104598615/5812/`, October 2009.

[5] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan. Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In *Proceedings of ER'09*, pages 428–443, Nov. 2009.

[6] F. Daniel, S. Soi, S. Tranquillini, F. Casati, C. Heng, and L. Yan. From People to Services to UI: Distributed Orchestration of User Interfaces. In *Proceedings of BPM'10*, pages 310–326., 2010.

[7] F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul. Understanding UI Integration: A survey of problems, technologies. *Internet Computing*, 11(3):59–66, May/June 2007.

[8] V. Hoyer and M. Fischer. Market Overview of Enterprise Mashup Tools. In *Proceedings of ICSOC*, 2008.

[9] A. Jhingran. Enterprise information mashups: integrating information, simply. In *Proceedings of VLDB'06*, 2006.

[10] R. Krummenacher, B. Norton, E. Simperl, and C. Pedrinaci. SOA4All: Enabling Web-scale Service Economies. In *Proceedings of ICSC2009*, 2009.

[11] D. Lizcano, J. Soriano, M. Reyes, and J. J. Hierro. EzWeb/FAST: Reporting on a Successful Mashup-based Solution for Developing and Deploying Composite Applications in the Upcoming Ubiquitous SOA. In *Proceedings of iiWAS2008*, Linz, Austria, 2008.

[12] T. Nestler, M. Feldmann, G. Hï£¡bsch, A. Preussner, and U. Jugel. The ServFace Builder - A WYSIWYG approach for building Service-based Applications. In *Proceedings of ICWE'10*, 2010.

[13] C. Pautasso. Composing RESTful Services with JOpera. In *Proceedings of SC'09*, pages 142–159, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] W.M.P. van der Aalst and K. van Hee. *Workflow Management - Models, Methods, and Systems*. The MIT Press, Cambridge, Massachusetts, 2002.

[15] J. Wong and J. I. Hong. Making Mashups with Marmite: Towards End-User Programming for the Web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007.

[16] L. Xie, P. de Vrieze, and L. Xu. When Social Software Meets Business Process Management. In *Proceedings of ICCIT2009*, 2009.

[17] G. O. Young. *The Mashup Opportunity*. Forrester Report, 2008.