A Mashup-Based Application for the Smart City Problematic

Abdelghani Atrouche[1(✉)], Djilali Idoughi[1], and Bertrand David[2,3]

[1] Laboratoire de Mathématiques Appliquées-LMA,
Université A. Mira, Bejaia, Algerie
{atrouche.a, djilali.idoughi}@gmail.com
[2] Université de Lyon, CNRS, Lyon, France
[3] École Centrale de Lyon, LIRIS, UMR5205, Écully, France
bertrand.david@ec-lyon.fr

Abstract. A mashup is an application that combines data and functionalities from more than one source. It groups disparate data in ways that enable users to do new things or accomplish common tasks with newfound efficiency. The introduction of mashup applications and their increasing use by users in the field of e-Learning and e-commerce highlights new issues in a context called the "smart city". Indeed, transportation based on private cars, public transportation services and shared bicycles need appropriate user interfaces, which can be "mashuped" to allow an integrated approach to transportation related to weather conditions, real-time traffic situations and personal preferences. These new needs for composition and combination (orchestration) of existing web services and their underlying user interfaces are good examples of mashuping. First, we provide in this paper some valuable explanations on two kinds of orchestration: service orchestration and HCI (Human Computer Interface) orchestration. Secondly, we apply this global approach to the context of "smart cities".

Keywords: HCI · Mashup · Smart city · Orchestration · Service orchestration · User interface orchestration

1 Introduction

Today, the main challenge facing software developers is to cope with development complexity and adaptation to frequent changes. The practice of software engineering is particularly essential for developing applications which are human life oriented. The "software design component-based" approach [1] is a valid approach, even for non-interactive software. For interactive software, the HCI dimension (Human Computer Interface) is required, especially in order to be able to adapt the interface to user demand (services) when he/she is dealing with software using different interaction devices. The HCI should provide better utilization, more adapted to user context features. To reflect the behavior of the system and its ease of use and availability, we must design these user interfaces with the same rigor and concerns as the systems themselves. The user interface must allow usability of the features expected by the user in

M. Kurosu (Ed.): Human-Computer Interaction, Part II, HCII 2015, LNCS 9170, pp. 683–694, 2015.
DOI: 10.1007/978-3-319-20916-6_63

different conditions of use and different interaction devices, naturally also in mobility. Furthermore, the composition of services is related to elaboration of an appropriate HCI, allowing users to handle these new services by a well-organized orchestration. The introduction of ICT (Information and Communication Technology) in the daily life of citizens has led to the emergence of new appropriate economic, social and environmental needs. In this context, the goal of the smart city, a major application field, is to allow citizens to be smarter, i.e. to allow them instant and quick access to services proposed by the city such as energy, transportation, culture, sport, etc.

In this paper, we propose a global approach for designing and implementing interactive software, integrating non-interactive components within a logic of orchestration of services and human-computer interfaces in accordance with user tasks. This takes the form of providing their generation and composition (mashup) by orchestration of the HCI.

2 Mashup Applications

Mashups are defined as "the perspective of software engineering. A mashup is constructed by the assembly and the combination of several existing functions integrated into a new application" [2].

The term "mashup" was defined initially in the field of music, where it consists of remixing two (or more) sounds in order to obtain a new one. Mashup is primarily and usually performed for the so-called "drag&drop" applications from different sources. The mashup architecture is made up of three elements according to Merrill [3]: Data, Services and User Interface. Mashup aims at the composition of a three-tier application: (1) Data (data integration), (2) Application logic (process integration) and (3) User interface (presentation integration). Integration of heterogeneous data sources uses two main technologies: web services and Mashup. Integration implies that all relevant data for a particular bounded and closed set of business processes is processed in the same software application.

Moreover, updates in one application module or component are reflected throughout the business process logic, with no complex external interfacing. Data are stored once, and are instantaneously shared by different business processes enabled by the software application [3]. In the Mashup, every user can compose his/her own service with other services in order to create a new service. Mashup is a "Consumer Centric Application". The Mashup describes web 2.0 sites combining functions of one site with another site. Different pieces of UIs (User Interfaces) are integrated into a new web application. This approach requires composition and orchestration of web services.

The goal of service composition is to produce richer applications. In UI integrations, it is very important to have a component model that can support interactions and compositions. When the user has the privilege of editing the Mashup, he/she is able to construct multiple applications and multiple versions of Mashups.

3 State of the Art

The concept of mashups has been popularized in the web application domain as a result of a large number of well-known web-based systems, such as Google Maps (maps.google.com) and Flickr (www.flickr.com). Their main contribution was to release their APIs to the public, thus enabling developers to leverage existing web technologies and create new applications. By restricting the development process to feature composition, mashups enable developers to rapidly create custom applications aimed at niche audiences [4, 5]. Web mashups [6] have turned lessons learned from data and application integration into lightweight, simple composition approaches featuring a significant innovation: integration at UI level.

Besides web services and data feeds, mashups reuse pieces of UIs (e.g., content extracted from web pages or JavaScript UI widgets) and integrate them into a new web page. Mashups, therefore, show a need for reuse in UI development and suitable UI component technologies.

The web mashup [7] phenomenon produced a set of mashup tools, aiming at assisting mashup development by means of easy-to-use graphical user interfaces targeted also at non-professional programmers. It is convenient to separate the types of mashups into the following three categories: data, user interface (UI), and process. Data mashups combine two or more data sets to create a new data set. UI mashups combine familiar UI elements to create new applications. Process mashups combine two or more processes into a single execution. It is also worth noting that most mashups fall into more than one of these categories or can be implemented in different ways.

As the orchestration HCI has proved very useful in service composition, it is important to provide some more details on orchestration. Orchestration is seen as "a musical arrangement to involve several instruments and therefore their process" [8]. It is thus an agreement or an organization defined by process management. According to SOA, orchestration is a mechanism that defines the integration or composition operation of services. On the other hand, there is an approach, namely user oriented orchestration, where caution is required when using the HCI.

Web services and SOA (Service Oriented Architecture), viewed in a process-oriented perspective, need a particular language to define how services can be composed into business processes. Such definitions would allow abstract processes as well as executable processes to be described [9].

In most service orchestration approaches, such as BPEL [10], there is no support for UI design. Many variations of BPEL have been developed, e.g., aiming at invocation of REST services [11] or at exposing BPEL processes as REST services [12]. In terms of standard BPEL, orchestration is a process made up of a set of associated activities (e.g., sequence, flow, if, assign, validate, or similar), variables (to store intermediate processing results), message exchanges, correlation sets (to correlate messages in conversations), and fault handlers. As UI orchestration has proved very useful in service composition, it is important to provide some more details on these two orchestrations: Service orchestration and UI Orchestration.

It is generally accepted that the architecture of a mashup is divided into three layers:

- Presentation/user interaction: this is the mashup UI. The main technologies used are HTML/XHTML, CSS, Javascript, Asynchronous Javascript and Xml (Ajax).
- Web Services: the product functionality can be accessed using API services. The main technologies used are XMLHTTPRequest, XML-RPC, JSON-RPC, SOAP and REST.
- Data: handling data like sending, storing and receiving. The main technologies used are XML, JSON and KML.

Architecturally, there are two styles of mashups: Web-based and server-based. Whereas Web-based mashups typically use the user's web browser to combine and reformat data, server-based mashups analyze and reformat data on a remote server and transmit these data to the user's browser in its final form.

To contribute to orchestration studies, we decided to increase the number of layers to five to explicitly introduce the Service Orchestration Layer and the UI Orchestration Layer.

4 Smart City

Smart Cities refer to cities that aim to improve the living standards of citizens - not only by developing new physical infrastructures, but also by using information technology. These solutions bring services closer to the community and play a positive role in an environmental context, thus helping to build sustainable cities. A smart city is characterized by a set of ICT applications corresponding to a wide range of issues related to traffic, energy, urban mobility, security, etc. For all these applications, we can distinguish end-users i.e. citizens, visitors, and operators i.e. employees.

Citizens and visitors are able to use these ICT applications to make everyday decisions based on processed information which could not be used manually. These applications use various data sources to present a high-level view of current events and offer decision guidance to users. A combination of these applications in coherent and useful proposals seems understandable. This is the first situation requiring mashup of these applications. City employees or companies are in charge of operating these city infrastructures, such as a bus company or rental bikes, in order to allow end-users to use them. They also need to acquire some integrated and specialized applications in a wide range of fields such as town planning, optimization and organization of transportation, civil services, waste disposal and so on, which are, however, limited in the operational scope to their responsibilities. The Mashup approach is an answer to these requirements. To concretize our explanation, we describe in the next section a scenario issued from our smart city application field.

4.1 A Case Study or Scenario

To explain the mashup approach we describe a scenario in which citizens as end-users and employees as operators use five services. They are mainly oriented to transportation

choices and related to observed situations made up of weather conditions, real-time traffic situations in public transportation and street circulation. These five services are as follows:

1. *Weather situation* is the service informing end-users in the short-term (less than 12 h) of weather conditions, allowing them to choose an appropriate means of travel.
2. *City atmospheric pollution service* provides users with the expected level of pollution for the same period.
3. *Bicycle rental fleet management service*, such as Vélo'V in Lyon, offers users the possibility to rent a bicycle with indication of availability in departure and arrival stations.
4. *Real-time vehicle traffic information service* provides overall city traffic information, as well as, more precisely, the local traffic situation (for a district).
5. *Real-time public transportation information service* is in charge of indicating bus, tram and metro traffic situations.

In our scenario, we have five web services: **weather application**, **traffic application**, **pollution application**, **management of rental bicycles** and **transportation management**. We use the UML Use case diagram to specify them (Fig. 1). Citizens use these services, and Operators (employees) are in charge of updating the status of these services.

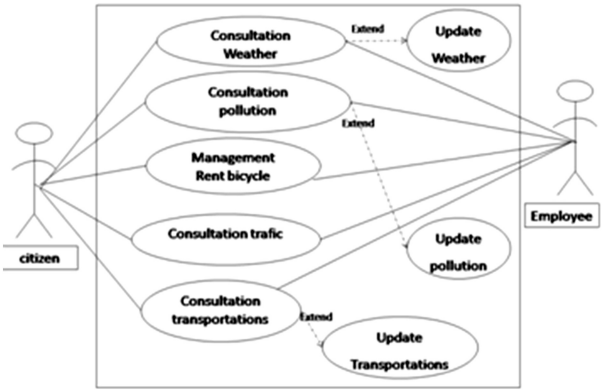

Fig. 1. Use case diagram of our Smart City scenario

These services are made up of several tasks which are either operator or end-user oriented. Operator-oriented tasks are working condition-oriented (such as indication of a new traffic jam area), while end-user tasks are use-oriented (to identify traffic jam areas). In both categories, more or less sophisticated tasks are provided such as simple observation of public transportation situation or appropriate trip structure between two points.

These services are, of course, Web services. They are available on various electronic devices located anywhere (personal apartment, public building, in street terminals and also and mainly personal devices i.e. tablets and smartphones). The main structure of these services is based on three parts: (1) specific application data, (2) SOA-based logical behaviors, and (3) user interface. However, the tasks proposed by each service are not available to everybody. Certain tasks are only operator-oriented, while others are end-user-oriented.

The first level of use is based on individual task (mono task) activation by the user interface and relates to a particular web service and corresponding User Interface. For example in Fig. 2, the end-user mono-task is weather consultation and the operator mono-task is weather update. In this relatively easy form of utilization, the application is not necessarily monolithic. It can use the mashup approach to indicate on the Google map the location of public transportation stations or traffic jam situations.

The second level of use is concerned with an integrated use of different tasks (called mashuped tasks) from different services in an integrated way; by interconnecting them with the mashup technology and orchestration of user interfaces. In Fig. 2, the end-user mashuped task aims at determining appropriate transportation in relation with weather and pollution conditions, street traffic and public transportation situations and bicycle availability. The Operator mashuped task allows him/her to modify public transportation in relation with increased pollution. At this higher level integration, the system is able to take into account weather conditions, atmospheric pollution, traffic conditions and user preferences (if they are available) in order to choose appropriate mono-mode or inter-mode transportation (same mode for all segments or different modes) (Fig. 2).

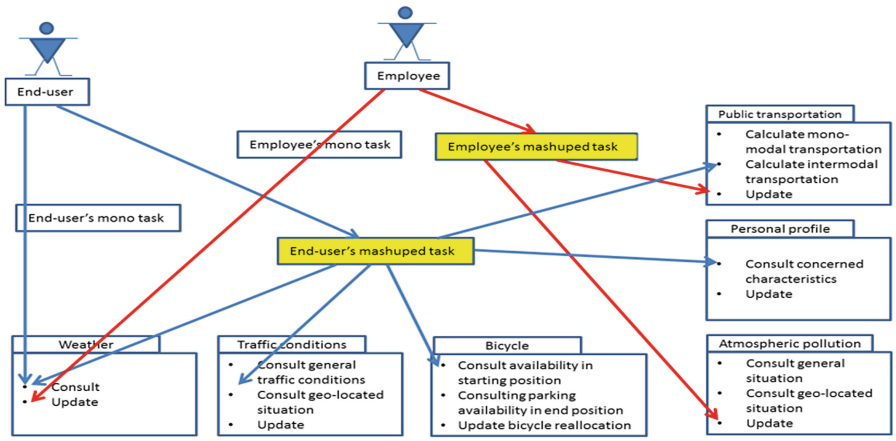

Fig. 2. End-User and Employee mono-tasks and mashuped tasks

5 Our Approach

5.1 Proposed Architecture

The architecture (Fig. 3) that we propose consists of five layers:

1. **User Interface:** the nearest layer to the user that presents to the user the final interface on his/her screen.
2. **UI Orchestration:** in this layer the UI orchestrator composes or integrates UI components allowing appropriate visualization and manipulation by users.
3. **Service Orchestration:** the layer where the Service orchestrator interconnects several services and their data in order to provide an appropriate functional answer as expected by the application. This interconnection is either a simple composition or a reasoning approach based on algorithm or knowledge manipulation.
4. **Services:** at this level all services are available and can access corresponding data sources located at the Data layer.
5. **Data:** this layer contains all sources of data needed by different services.

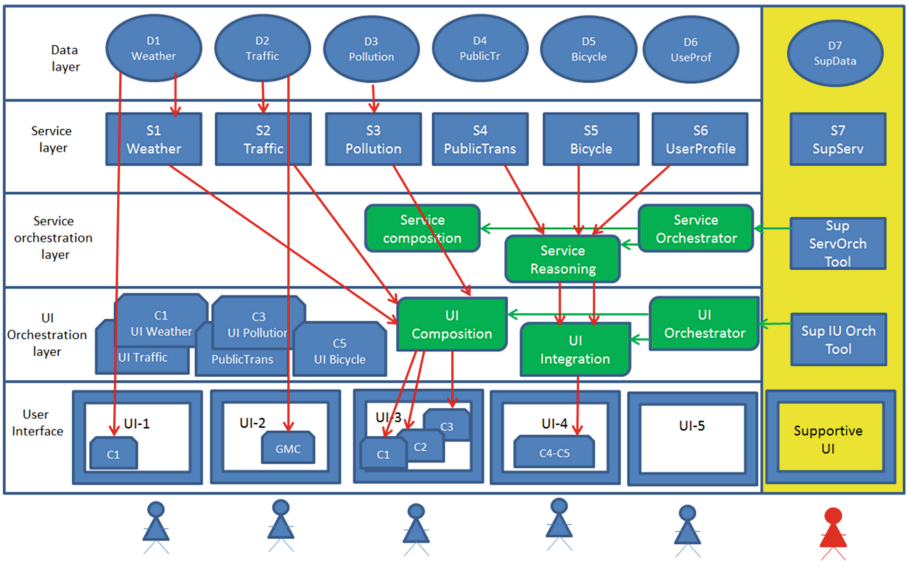

Fig. 3. The multi-layer mashup-based application architecture

On Fig. 3, we show an overall architecture model with 5 layers broadly explained hereafter with typical mashup situations related to our scenario:

- User interface 1 (UI-1) explains a classical situation where a UI component (widget) shows the current weather situation on the screen. The corresponding weather service task delivers current data.

- User interface 2 (UI-2) corresponds to the first mashuping situation, in which traffic situation data are not visualized as a list, but mashuped in a Google Map Component (GMC). In this case, application data are distributed geographically on the city Google Map.
- User interface 3 (UI-3) is in charge of delivering on the same screen information provided by different services. UI orchestration is easy, based only on composition of different UI components (widgets). Each widget is in relation with one task of the corresponding service, which is in charge of delivering appropriate data. In our case the user can observe at the same time weather, traffic and pollution situations.
- User Interface 4 (UI-4) presents a more sophisticated situation. The goal is to collect information concerning user profile, public transportation situation and bicycle availability in the departure station and space availability at the destination in order to offer the user an intermodal transportation based on several public transportation segments (bus, tram and metro) combined with bicycle segments (if the user profile is compatible with its use). For this application we have grouped all proposed aspects. We need to mashup different services in order to determine a composite transportation trajectory compatible with the user profile. Then we must find an appropriate presentation based on integration of different widgets. This means, at the service orchestration layer, that service orchestration must propose service reasoning able to find appropriate transportation segments with their transportation supports (bus, tram, metro or bicycle). It must then transmit this information to the UI Orchestration layer, where UI Integration must be provided to allow appropriate information presentation. At this layer, integration between public transportation widgets and the bicycle widget must be provided.
- The last case, which is not materialized (UI-5), is concerned with mashup of all services: choice of transportation in a composite intermodal way using, potentially, private car, several public transportation forms (bus, tram and/or metro) and bicycle in relation with user profile (age, bicycle ability, etc.), traffic situation, weather conditions and pollution. This service mashuping is reasoning- and/or algorithm-oriented in order to determine appropriate transportation segment scheduling according to the current situation (weather, pollution, traffic and public transportation conditions and user preferences. Once the transportation means has been determined, it is important to show chosen transportation schedule data appropriately. For this purpose, an integrated UI based on integrated widgets is needed.

In our architecture, two main layers are concerned with orchestration and mashuping: the **Service Orchestration layer** and the **User Interface orchestration layer**.

5.2 Service Orchestration Layer

At this layer, the **service orchestrator** is in charge of orchestrating services either by composition or by reasoning. Composition aims at unifying different services through seamless functional integration of aggregated sources. The resulting mashup is a new application, and participating sources are components of this new application.

Reasoning refers to integration of aggregated resources and services, which are connected by logical (reasoning rules) or algorithms in order to create new aggregated functionalities.

5.3 User Interface Orchestration Layer

At this layer, the **UI orchestrator** is in charge of orchestrating UI interfaces either by composition or by integration. The goal of UI composition is to present on the screen a collection of independent basic UI components (widgets). UI integration is concerned with integration of several UI components (widgets) into a single one.

5.4 Scenario-Based Explanations

Let us now show the role of these two orchestrators on our scenario situations:

1. UI-1 is a standalone application made up of Weather service (S1) and a UI-component showing and managing Weather visualization. The Service Orchestration layer and the UI orchestration layer are not concerned.
2. UI-2 is a Traffic management application made up of Traffic service (S2) and a UI component, Google Maps, allowing visualization of traffic data (situation) in a map perspective. This is what we call vertical mashup between a service and an open visualization application able to take into account the data to be projected on the map.
3. UI-3 is a mashuped application taking into account multiple data delivered by three services (Weather – S1, Traffic – S2 and Pollution – S3). At the service orchestration layer, no action is required: these services deliver independent data and propagate them to the UI Orchestration layer, where the UI Orchestrator uses UI composition to show on the screen three UI components, visualizing independently these data on a single surface.
4. UI-4 is a more sophisticated situation in which two orchestration layers are required. At service orchestration level, the service orchestrator asks Service Reasoning to group User Profile information, the Public Transportation situation and Bicycle management, treated by services S6, S4 and S5, respectively. Service reasoning is the service mashup which, in relation with user profile, indicates user ability to use a bicycle and elaborates a transportation schedule either mixing public transportation and bicycle segments or using public transportation only. The UI Orchestrator at the UI orchestration layer is in charge of integrating UI components C4 & C5 to show the trip structure.
5. UI-5 is, as described, a sophisticated mashuping at the service orchestration layer with weather, pollution and user profile conditions to determine the appropriate transportation solution based on an inter-modal aggregation. The optimized trip structure is presented in a mashuped single UI component showing the user the proposed trip.

It is important to observe that, at the Service Orchestration layer, mashuping is based on the study, connection and interrelation between different tasks available for each service. At the UI orchestration layer, the main goal is to deliver to each user in respect to his/her role, the appropriate user interface for both visualization and interaction. The buttons, menus and commands proposed are in relation with the user's accreditations, i.e. use for citizens, and management for city operators. The UI orchestration layer is in charge of managing task authorization and filtering.

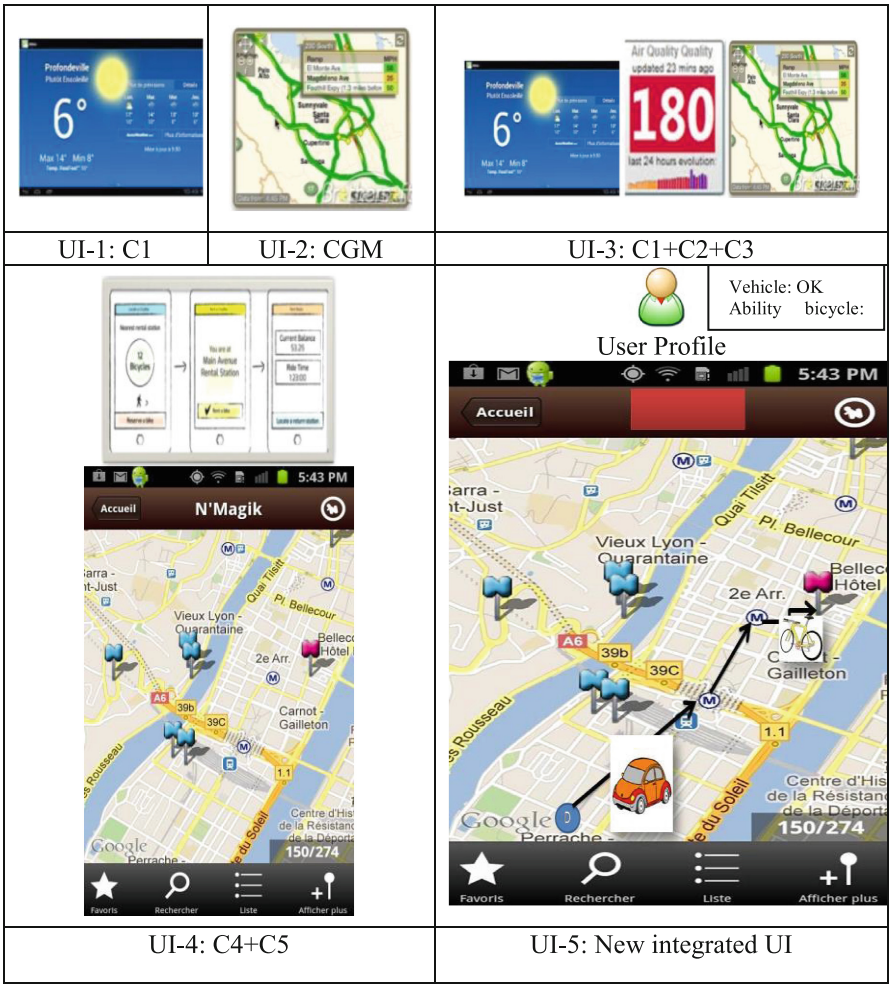

Fig. 4. Large screen information scenarios concerning transportation proposals

5.5 Supportive User Interface

On the right side of Fig. 3 we show a new module added to the overall architecture model with 5 layers, as explained previously. This module, called the supportive User Interface [13], is in charge of creating new mashuped applications. Supervision data, supervision services, the Supervision Service Orchestration Tool and the Supervision UI Orchestration Tool are the main components of this module. The supportive User Interface offers users different possibilities for manipulation as well as facilitating manipulation of different services. Two working conditions are proposed: programming devoted to trained developers and a visual programming-oriented approach in which a light version of mashuping is provided to less trained developers and high-level users. Naturally, at the Service Orchestration layer, visual programming, i.e. graphical manipulation of concepts, is mainly used to compose tasks and data and to express relatively simple and easily expressed reasoning. At the UI Orchestration layer, composition of UI components (widgets) is supported as well as the low level of UI Integration. Naturally, the user-friendliness of this interface is the decisive factor for determining utilization: only by professional developers or also by experienced users (Figs. 4 and 5).

Fig. 5. Smartphone UI scenario

6 Conclusion

Application mashuping is a main and much appreciated approach for creating appropriate applications based on reuse of existing ones. In this paper, we tried to clarify this large and flourishing activity by means of multiple solutions and approaches. As our application field we took the smart city and its need to receive appropriate applications which can be elaborated by applying the mashuping approach. We proposed an architecture based on five layers, allowing both clear distinction of data, services and UI, as well as, with respect to mashuping, the Service orchestration layer and the UI orchestration layer. We consider this separation to be essential as Service orchestration and UI orchestration are very different. We explained the main mashuping techniques

used at these two layers: composition and algorithmic or reasoning for service orchestration, and UI composition and UI integration for UI orchestration. We also proposed à supportive user interface, which is mashup application creation- or evolution-oriented, either with a programming interface for experienced developers or a visual programming-oriented approach for experienced users.

We are currently looking into the possibility of using this supportive visual programming- oriented interface while using mashuped applications in order to adjust their behaviors.

References

1. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development and its differences with traditional integration. IEEE Internet Comput. **12**(5), 44–52 (2008)
2. Lizcano, D., Soriano, J., Reyes, M., Hierro, J.J.: EzWeb/FAST: reporting on a successful Mashup-based solution for developing and deploying composite applications in the upcoming ubiquitous SOA. In: Proceeding of the Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM 2008, Valencia, Spain, pp. 488–495 (2008)
3. Merrill, D.: Mashups: the new breed of Web app: an introduction to mashups (2009). http://www.128.ibm.com/developerworks/xml/library/x-Mashups.html
4. Grammel, L., Storey, M.: An end user perspective on mashup making, University of Victoria Technical Report DCS-324-IR (2008)
5. Hartmann, B., Doorley, S., Klemmer, S.: Hacking, mashing, gluing: a study of opportunistic design and development. Pervasive Comput. **7**(3), 46–54 (2006)
6. Caste, B.: Introduction to Web Services for Remote Portlets, IBM Developerworks (2005). http://www-128.ibm.com/developerworks/webservices/library/ws-wsrp/
7. Beinhauer, W., Schlegel, T.: User Interfaces for Service Oriented Architectures, July 2005. http://www.webservice-kompass.de/fileadmin/publikationen/User_Interfaces_for_Service_Oriented_Architectures.pdf
8. David, B., Chalon, R.: Orchestration modeling of interactive systems. In: Jacko, J.A. (ed.) HCI International 2009, Part I. LNCS, vol. 5610, pp. 796–805. Springer, Heidelberg (2009)
9. Erl, T.: Service Oriented Architecture, Concepts, Technology, and Design. Prentice Hall, Upper Saddle River (2006)
10. OASIS. Web Services Business Process Execution Language Version 2.0, April 2005. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
11. Pautasso, C.: BPEL for REST. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 278–293. Springer, Heidelberg (2008)
12. Van Lessen T., Leymann F., Mietzner R., Nitzsche J., Schleicher D.: A management framework for WS-BPEL. In: ECoWS 2008, Dublin, pp. 187–196 (2008)
13. David, B., Chalon, R., Delomier, F.: Supportive user interfaces for MOCOCO (mobile, contextualized and collaborative) applications. In: Kurosu, M. (ed.) HCII/HCI 2013, Part V. LNCS, vol. 8008, pp. 29–38. Springer, Heidelberg (2013)