End-User Development of Mobile Mashups

Cinzia Cappiello, Maristella Matera, and Matteo Picozzi

Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
via Ponzio, 34/5, 20133, Milano, Italy
{cinzia.cappiello,maristella.matera}@polimi.it,
matteo.picozzi@elet.polimi.it

Abstract. The spread of mobile devices empowers more and more end users to access services publicly available on the Web. It also encourages users to construct applications satisfying their situational needs, by customizing and combining the huge amount of online resources. *Mobile mashups* have the potential to accommodate this trend, providing a flexible paradigm for a service-based development of mobile applications. This paper introduces and End User Development (EUD) framework, based on a model-driven approach for the design and the automatic generation of mobile mashups. The approach is characterized by a "lightweight" composition paradigm that exploits visual notations for the specification of data integration and service synchronization rules.

Keywords: Mobile Mashups, End-User Development, Model-Driven Mashup Development, Data Integration, Data Fusion.

1 Introduction

With the evolution of the mobile world and the proliferation of available services, *Mobile Internet* is becoming a reality that empowers users to access services pervasively. This scenario constantly increases the desire of users to participate in the development of their own artifacts. This trend was already initiated by the Web mashup paradigm, through which users have started to be actively engaged in the creation of Web applications, to satisfy the "long-tail" of niche functions not always addressed by the most common applications [1]. The need to self-create applications is now even stronger for the mobile users, for which simple applications addressing very contingent requirements, as mashups are, can solve several information needs arising in mobile usage contexts.

As also demonstrated by the last years' research on Web mashups, only few users are able to create their applications by programming the service integration. Difficulties also arise when using some tools for visual composition that however do not adequately abstract from the underlining technologies and languages [2,3]. This situation is even accentuated in the mobile context, where also expert users would need to get experienced with very specific technologies that depend on the target mobile device. Composition paradigms and tools, abstracting from implementation details, would therefore really help users to construct their applications.

To address the issues illustrated above, this paper proposes a framework for the lightweight, user-driven composition and deployment of *mobile mashups.* The target of our composition approach are device's native applications that, in contrast with Web mashups, do not need the Web browser as execution environment, and thus facilitate the access to mobile device services. One notable feature of our composition framework is the intuitiveness and ease of use of the composition language, based on visual mechanisms that allow the users to easily configure the fusion of contents coming from different data sources, and the synchronization of such core contents with further UI-rich remote components [4] and services local to the mobile device.

The paper is organized as follows. Section 2 summarizes some related works in the field of user-created mobile apps and mobile mashups. Section 3 describes the overall organization of our platform, and also illustrates our approach for visual-oriented data integration and service integration. Section 4 finally draws our conclusions and outlines our future work.

2 Rationale and Background

Mashups have been originally conceived in the Web context, to collect data and functions from different sources and construct integrated Web applications. The mashup flexibility, deriving from the possibility to compose varying services, combined with the diffusion and capabilities of mobile devices, is now leading to an increased interest in a new *mobile mashup* paradigm [5,6], whose potential is to respond to the specific user needs hidden in the long tail of requirements not satisfied by existing applications. Indeed, although the capability of current mobile devices to access services, compute data and display results, providing the users with the right information in the right moment is still an issue for several mobile apps.

Mobile mashups are the object of our research. We define mobile mashups as *native mobile apps*, constructed by integrating data coming from different services and/or synchronizing services that can be remote (e.g., available in the Web), or local (i.e., available on the mobile terminal). Our approach to mashup development in particular focuses on End-User Development (EUD) practices [3,1], trying to provide solutions to some open issues in the composition of mobile mashups by the end users. Mobile mashups currently in use are indeed rigid, hard coded applications, produced by expert developers by programming the service invocation and the service integration logic.

Mobile End-User development is still in its infancy and presents new issues mainly related to the need of identifying composition paradigms and models for the final apps that can fit the characteristics of the mobile devices and of the mobile context of use [7]. Some approaches offer authoring environments directly on the mobile devices; however, also due to the well-know limitations of the terminal devices, the mobile editors just enable the production of very simple applications, so-called *micro-services*, very limited in terms of offered contents and functionality [8,9]. Such authoring approaches, although based on intuitive visual metaphors, do not address at all the composition of remote services and APIs. The TELAR platform [10] goes some steps further and assists the construction of map-based Web mashups, deployed as HTML pages enriched with JavaScript code running in the mobile Web browser. TELAR in particular enables the integration of selected data sources with Google Maps, but the only flexibility shown by this approach is the ability of the mashup client (the mobile web browser) to load an XML file when the HTML page including the mashup is downloaded, which includes configuration parameters referring to the data sources to be queried. The platform does not offer any support for the specification of the configuration parameters - in case of changes, the user is thus required to write XML code and upload it on the mobile device.

An interesting approach is reported in [11], where the authors present a platform for mashup composition where the users can configure, through a form-based paradigm, some query parameters for service invocation. The limit of this approach is however that service integration is possible only between a selected service and some *zone maps*. Thus the approach is exclusively oriented to the construction of mashups for the access to zone-based services within interactive public spaces (e.g., within a mall).

In [12] the authors illustrate a mobile generator system that, in line with our perspective on the user-centric development of mobile mashups, offers a desktop environment that automatically generates the code of the mobile application. The composition paradigm however shows an intrinsic complexity, being it based on the specification of parameter values and programming scripts. Also, while this approach enables content extraction from Web documents, it does not support at all data and service integration. We believe this is instead a fundamental feature for the mobile usage context, where the access to integrated views of the retrieved contents can definitely improve the user experience and productivity.

3 Approach and Architecture

In our approach for the composition of mobile mashups, the end users, by means of visual actions and guided by *visual templates*, are enabled to construct unified views over integrated data sets extracted from multiple sources, (e.g., music events published by REST services, such as Upcoming and Brite), and to synchronize items from such data views with UI-rich components (e.g., tapping on a geo-referenced music event triggers the localization of that item on Google Map) and with services local to the mobile device (e.g., tapping on a phone number triggers the store of that number in local device agenda). The design activity is supported by a Web-based design environment that offers a composition paradigm strongly characterized by a visual approach: i) users are required to perform only visual actions to compose their applications, and ii) any concept in the underlying mashup model has a visual characterization in the composition paradigm. The visual nature of our approach is reflected by the notion of *visual templates*, i.e., a set of abstractions that constitute the interface between the visual composition paradigm requiring the completion of user interface skeletons with data and functions extracted from services, and the underlying mashup model for data and service integration.

3.1 Visual Templates

Within the platform design environment Visual templates offer an immediate representation of the user interface through which the data set filtered out and integrated by the user composition actions will be displayed in the final application. The completion of visual templates with data items retrieved through remote services enables an "integration-by-example" paradigm, through which the users can (i) select data from pertinent data sources, and (ii) in case of multiple sources express how data have to be merged and displayed.

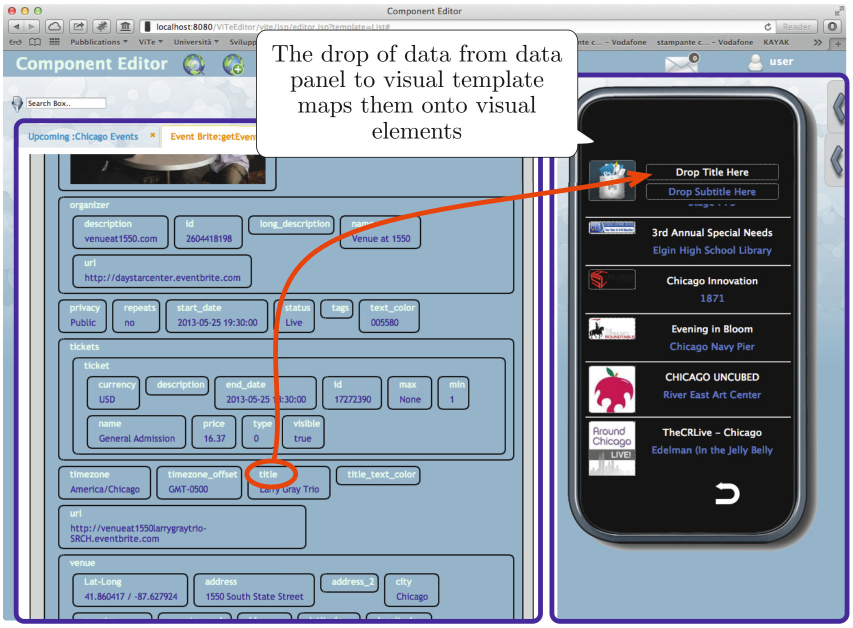
Figure 1 illustrates the design environment where the visual composition takes place. The left part of the screen, the *data panel*, is devoted to show the data retrieved by querying services. The right hand area, the *visual template panel*, instead shows one selected visual template. The visual template reported in figure is based on a list-based representation of data items. In our current platform prototype, also map-based and chart-based visual templates are available. In order to support our data integration paradigm (which we describe in the sequel of this section), the visual templates are organized into two main views. A *global view*, as the one reported in figure, visualizes all the retrieved data by means of the attributes selected by the user in the data panel. In a list of concerts, as the one represented in figure, each instance is for example represented though a title, a subtitle, and an image; on a map-based template POIs can instead identify the retrieved instances. A secondary view, which can be accessed on demand, then display further data attributes of one selected instance.

Independently of its concrete display (e.g., whether through a list, a map or a chart), a visual template can be seen as a tuple, $VT =< vr_1, vr_2, \ldots, vr_n >$, where each vr_k represents a *visual renderer* serving the visualization of a data attribute extracted by one of the user selected sources. Each visual template can be then decomposed in two sub-templates, representing the organization of the two different screens in the final application:

- A *union sub-template*, whose vr_k (i.e., uvr_k) display data attributes that concisely represent the instances retrieved by querying the involved services (e.g., for a concert, the name of the player, her music gender, a picture).
- A *merge sub-template*, whose vr_k (i.e., mvr_k) display the details of a data instance selected in the union sub-template. While the union sub-template has a fixed organization, the structure of the merge sub-template, i.e., the visual renderers included in it, is determined by the the number and type of the heterogeneous data elements that the user selects as details to be displayed for each item in the union sub-template.

3.2 Data Source and Service Integration

Given the availability of a set of registered data sources and of a visual template, the user constructs the presentation layer of the final application - and implicitly



organizer: description venueat1550.com; id 2604418198; long_description; name Venue at 1550; url http://daystarcenter.eventbrite.com

privacy Public; repeats no; start_date 2013-05-25 19:30:00; status Live; tags; text_color 005580

tickets / ticket: currency USD; description; end_date 2013-05-25 19:30:00; id 17272390; max None; min 1; name General Admission; price 16.37; type 0; visible true

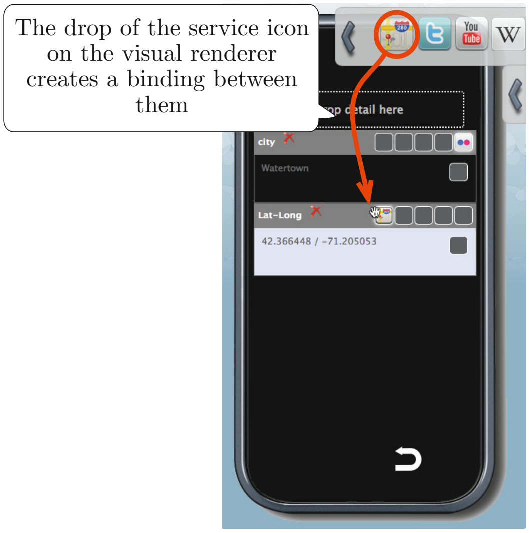
timezone America/Chicago; timezone_offset GMT-0500; title Larry Gray Trio; title_text_color

url http://venueat1550larrygraytrio-SRCH.eventbrite.com

venue: Lat-Long 41.860417 / -87.627924; address 1550 South State Street; address_2; city Chicago

Data panel Visual Template Panel

(a) Data panel and the global sub-template


The drop of the service icon on the visual renderer creates a binding between them
op detail here

city: Watertown

Lat-Long: 42.366448 / -71.205053

(b) Detail sub-template and the menu with some of UI rich components available in the platform

Fig. 1. The design environment where the user can query services and map the retrieved data onto elements of *visual templates*.

integrates data. If the user select data from different sources, a data integration process starts, and the user actions are therefore targeted towards the construction of a unified data view. The approach consists of phases comparable to those ones occurring in the classical data integration processes. However, each phase is conceived to make the overall process lightweight. Also, to better support the end-users, some phases are automatically performed by the design environment.

The data integration process starts with the *source data sampling* phase in which each selected data source is queried according to the basic parameters and settings defined at registration time. The returned result set is thus processed and visualized through an automatically generated representation of "attribute-value" items within the data panel (see the left area in Figure 1(a)). The user then *selects a visual template* as the basis for the visualization of data retrieved by the selected services. This selection implicitly corresponds to the *definition of the global schema* for data integration. In fact, at the end of the composition process, the visual template provides a unified schema that guides the extraction of data from each involved service.

A *visual mapping* follows, in which the user associates data fields visualized in the data panel, which are extracted from one or more data services, with some notable elements of the visual template, the so-called *visual renderers*. This association operates a reduction of the data service schema that is used at runtime for querying the service. In other words such visual mapping process constructs the integration queries, mapping the local schema of each single service to the global schema imposed by the visual template.

The data integration in our approach is based on two important assumptions: (i) the user can ask for the integration of two or more sources associated with the same visual template and (ii) the different sources are intersected on the basis of a subset of the fields contained in the union visual template (i.e., the title field in case of a list-based visual template, the geographical position in case of a map-based visual template). The global schema is built by considering the set of visual renderers included in the visual template. The mapping between the local service schema and the global integration schema is derived by the association between data attributes and visual renderers as defined by the user.

Besides the integration of data sources according to the visual template paradigm, the platform design environment also allows the user to synchronize the unified data view with other services. The aim is to enrich the core data with contents of different nature. For example, data on concerts could be integrated with multimedia contents (videos and images) retrieved through APIs such as YouTube and Flickr. In line with the Web mashup paradigm defined by our previous research [13,3], the design environment allows the users to bind the data attribute displayed through visual renderers with requests to operations offered by UI-rich APIs. Such APIs are registered into the platform and wrapped to make it possible invoking their functionality. As shown in Figure 1, the available APIs are shown to the user through an icon menu, which adapts itself showing the only APIs that are compatible with a given visual renderer. To create bindings, users move the API icon on a visual renderer. At runtime, the same icon, displayed next to the visual renderer, enables the invocation of the API. With the same mechanism, it is also possible to take advantage of the mobile device capabilities and bind visual renderers to device services such as the dialer service or the address book.

3.3 Generation of the Application Schema

Based on the visual actions for data integration and service binding definition performed by the user, the design environment generates and upload on a server a configuration file that contains rules for the automatic instantiation of the final mobile app. In the following, a (simplified) fragment of the application schema is showed; it refers to a mashup offering data about concerts and the access to UI components (Flickr, Twitter, Wikipedia) and device custom apps (dialer and address book).

```
<sources>
 <category name="Concerts" icon="...">
  <source name="EventBright" url="...">
   <params>
    <param name="location" label="Location" type="input">chicago</param>
    ...
   </params>
  </source>
  <source name="Upcoming" ...>...</source>
 </category>
</sources>

<visual-mapping>
 <union type="List">
  <vr name="Title" type="Text">
   <data source="EventBright" query="/title"/>
   <data source="Upcoming" query="/@name"/>
  </vr>
  <vr name="Subtitle" type="Text">
   <data source="EventBright" query="/venue/name"/>
   <data source="Upcoming" query="/@venue_name"/>
  </vr>
  <vr name="Image" type="Image">
   <data source="EventBright" query="/venue/image"/>
   <data source="Upcoming" query="/@photo_url"/>
  </vr>
 </union>
 <merge>
  <vr name="State" type="Text" source="EventBright" query="..."
      binding="twitter|flickr|wiki"/>
  <vr name="Telephone" type="Text" source="EventBright" query="..."
      binding="dialer|addressbook"/>
  <vr name="City" type="Text" source="Upcoming" query="..."
      binding="twitter|flickr|wiki"/>
  <vr name="Address" type="Text" source="Upcoming" query="..."
      binding="maps"/>
 </merge>
</visual-mapping>
```

The generated schema reflects our model for mobile mashups, and based on the structure of the visual template, it specifies:

– The *presentation layer*: it represent the *vr* elements as deriving by the visual template filled in during the mashup design. An attribute `type` for each *vr* denotes the type of visual element (for example, a text field or an image placeholder) that is automatically derived by the design phase, taking into account the type of the associated data item.

– The *service data layer*: for each *vr*, the `data source` attribute specifies the service providing the mapped data and the corresponded query; the setting to invoke the service (service URL and possible parameters) are expressed in the source specification, at the beginning of the configuration file. Source properties also include the definition of possible filters to be used during the app execution to progressively refine the service result set.
– The *service binding layer*: for each *vr* the `binding` attribute specifies the list of services subscribed to the selection of a data item. Since the management of this bindings is based on service wrappers, no other details about how the services must be queried are needed. The wrappers for the involved services will be downloaded on the mobile device together with the application schema.

3.4 The Device Execution Environment

Figure 2 illustrates the main elements composing the execution engine running on the mobile device. The *Rules Interpreter* parses the application schema created at design time. The users download the execution engine app from the same server where they download the mobile mashup schema. The execution engine

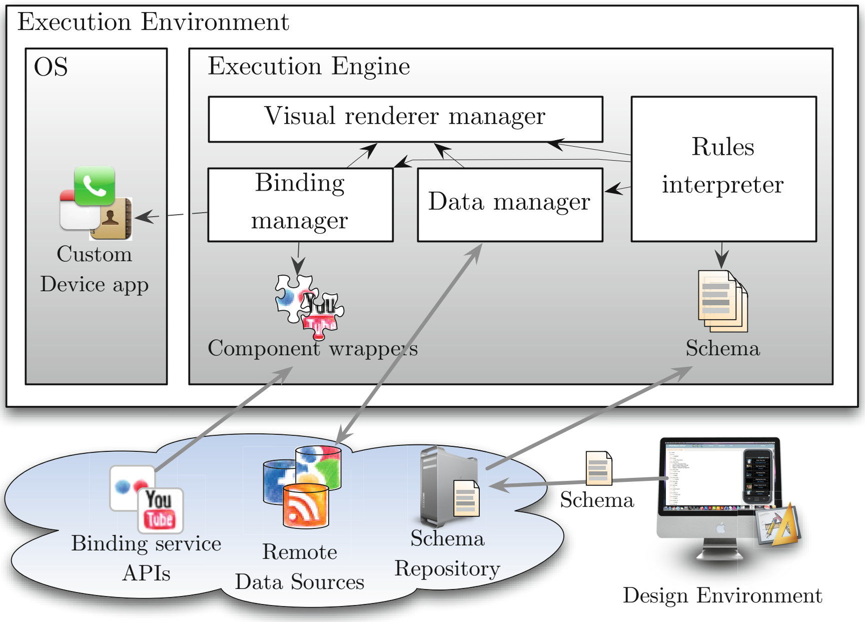


Fig. 2. Architecture of the execution environment

masters the creation of the application by invoking the other modules in charge of managing the different application layers. In particular:

– Based on the visual template definition, the *visual renderer manager (VRM)* translates the specified *vr* elements into native code for the UI layout generation. For example, if Android is the target OS, the application schema is translated into the Android layout markup language that will be used by the OS for the generation of screens – so-called *activities*. The VRM also handles the visualization of data into each *vr*. The VRM is also in charge of constructing dynamically other UI elements, for example the app menu.
– Based on the visual mapping rules defined for each *vr*, a *data manager (DM)* queries the involved services, according to the specifications of the *vr* `data source` attribute and the data source settings. The DM module also manages the detection and elimination of duplicates, by means of the Soundex similarity algorithm [14], and the fusion of corresponding data attributes into the merge sub-template.
– Based on binding rules, the *binding manager (BM)* sends a request to the VRM for the creation of the screens for displaying UI components. During the mashup execution, BM is then in charge of "listening" to the selection of *publisher* data items, capturing parameters embedded in such data items, and invoking, by means of the API wrappers, the synchronized service operations. The results are then sent back to VRM module.

4 Conclusions

In this paper we have introduced our perspective over the development of mobile mashups; thus we have discussed how adequate abstractions, mainly based on visual template construction, can enable a lightweight data integration process, leading to the definition of unified data views, and the synchronization between such data and remote APIs and custom device services. A formative evaluation session with real users has confirmed the effectiveness of such choices. Also, some performance tests, executed on a Samsung SII smart phone, have shown that our lightweight data fusion paradigm is possible in reasonable time on the mobile device. Although optimized for mobile mashups, we also believe that our approach can target as well other execution platform, with the only effort of identifying adequate visual templates and modifying the logics behind the app instantiation by the execution engine.

The examples shown throughout the paper refer to easy configuration of services and service integration. They also assume a set of default rules, for example related to the operations that can be invoked trough UI components. We however believe that with few extensions, but keeping the same visual metaphor, our current platform can evolve towards a development environment for expert users, giving them the possibility to configure services and UI components more flexibly, and also introducing more sophisticated conflict resolution policies in the construction of the unified data views. Our future work is devoted to improve the design environment along this direction, trying to achieve a full-fledged approach, accommodating different expertise levels.

References

1. Daniel, F., Matera, M., Weiss, M.: Next in mashup development: User-created apps on the web. IT Professional 13(5), 22–29 (2011)
2. Namoun, A., Nestler, T., De Angeli, A.: Conceptual and usability issues in the composable web of software services. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 396–407. Springer, Heidelberg (2010)
3. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: Dashmash: A mashup environment for end user development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
4. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding ui integration: A survey of problems, technologies, and opportunities. IEEE Internet Computing 11(3), 59–66 (2007)
5. Maximilien, E.M.: Mobile mashups: Thoughts, directions, and challenges. In: Proceedings of ICSC 2008, August 4-7, pp. 597–600. IEEE Computer Society, Santa Clara (2008)
6. Xu, K., Zhang, X., Song, M., Song, J.: Mobile mashup: Architecture, challenges and suggestions. In: Proc. of Management and Service Science, MASS 2009, pp. 1–4 (September 2009)
7. Cuccurullo, S., Francese, R., Risi, M., Tortora, G.: Microapps development on mobile phones. In: Piccinno, A. (ed.) IS-EUD 2011. LNCS, vol. 6654, pp. 289–294. Springer, Heidelberg (2011)
8. Häkkilä, J., Korpipää, P., Ronkainen, S., Tuomela, U.: Interaction and end-user programming with a context-aware mobile application. In: Costabile, M.F., Paternó, F. (eds.) INTERACT 2005. LNCS, vol. 3585, pp. 927–937. Springer, Heidelberg (2005)
9. Davies, M., Fensel, A., Carrez, F., Narganes, M., Urdiales, D., Danado, J.: Defining user-generated services in a semantically-enabled mobile platform. In: Kotsis, G., Taniar, D., Pardede, E., Saleh, I., Khalil, I. (eds.) iiWAS, pp. 333–340. ACM (2010)
10. Brodt, A., Nicklas, D.: The telar mobile mashup platform for nokia internet tablets. In: EDBT. ACM International Conference Proceeding Series, vol. 261, pp. 700–704. ACM (2008)
11. Soroker, D., Paik, Y.S., Moon, Y.S., McFaddin, S., Narayanaswami, C., Jang, H.K., Coffman, D., Lee, M.C., Lee, J.K., Park, J.W.: User-driven visual mashups in interactive public spaces. In: Cahill, V. (ed.) MobiQuitous. ACM (2008)
12. Chaisatien, P., Prutsachainimmit, K., Tokuda, T.: Mobile mashup generator system for cooperative applications of different mobile devices. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 182–197. Springer, Heidelberg (2011)
13. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. In: Proc. of WWW 2007, pp. 923–932 (2007)
14. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. 41(1) (2008)