# On the Role of Context in the Design of Mobile Mashups

Valerio Cassani, Stefano Gianelli, Maristella Matera, Riccardo Medana, Elisa Quintarelli, Letizia Tanca, and Vittorio Zaccaria

Politecnico di Milano
*Dipartimento di Elettronica, Informazione e Bioingegneria*
{valerio.cassani,stefano.gianelli}@mail.polimi.it, name.surname@polimi.it

**Abstract.** This paper presents a design method and an accompanying platform for the development of Context-Aware Mobile mashUpS (CAMUS). The approach is characterized by the role given to context as a first-class modeling dimension used to support the identification of the most adequate resources that can satisfy the users' situational needs and the consequent tailoring at runtime of the provided data and functions.

**Keywords:** Mobile Mashups, Mashup Modeling, Context Modeling, Context-aware Mobile Applications, GraphQL

## 1 Introduction

The data deluge we are confronting today takes everyone to continuously search and discover new information. The opportunity to access a large amount of information, however, does not always correspond to an increase of knowledge. Many times, indeed, one does not know how to filter data "on-the-fly" to obtain the information that is the most suitable to the current context of use. This is even more critical when using mobile devices that are still characterized by limited capabilities (memory, computational power, transmission budget).

Given this evidence, our research focuses on the definition of methods and tools for the design and development of *Context-Aware Mobile mashUpS* (CAMUS)[1] that are based on a set of high-level abstractions for context and mashup modeling. In particular, we have defined a novel design methodology and related tools for fast prototyping of mobile mashups, where context becomes a first-class modeling dimension. In comparison to other approaches to mashup design [2], the composition activity and, more specifically, the selection of services are not exclusively driven by the functional characteristics of the available services or by the compatibility of their input and output parameters. Rather, the initial specification of context requirements enables the progressive filtering of services first and then the tailoring of service data to support the final situations of use.

## 2 Methodology

Figure 1 represents the general organization of the CAMUS design framework and highlights the flow of the different activities and related artifacts that en-

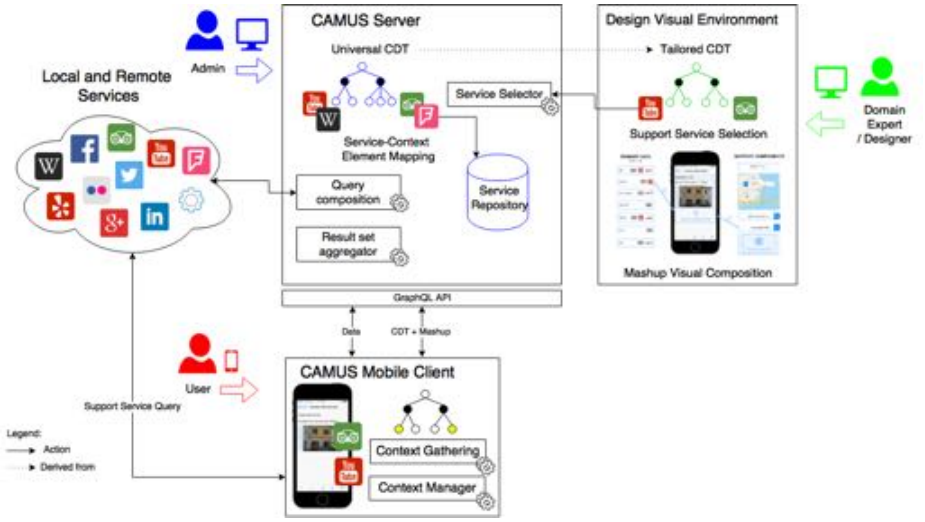


**Fig. 1.** Organization of the CAMUS framework, highlighting the main system components and the supported design and execution activities.

able the transition from high-level modeling notations to running code. The first step in the design process is the specification of context requirements. All the aspects that characterize the different contextual situations, i.e., the *dimensions* contributing to context, are represented by means of the so-called *Context Dimension Model* [3], which provides the constructs to define at design-time the Universal Context Dimension Tree (*Universal CDT*), i.e., the set of possible contexts of use for a given domain of interest, expressed as a hierarchical structure (see Figure 2 for an ecample). Then different activities follow as reported in the following.

**Creation of the Service Ecosystem.** The *platform administrator* is in charge of managing the CAMUS server. One of the main roles is to create and maintain the *service repository*. S/He registers distributed resources (remote APIs or in-house services) into the platform, by creating descriptors that specify

- *How the resources are to be invoked*, e.g, the service endpoint, its operations and input parameters. In this phase, some parameters can be bound to wrappers that perform transformations from symbolic context values gathered at runtime to corresponding numerical service input.
- *The schema of the returned service responses*. To ensure homogeneity of data formats, needed to merge the data that must be visualized by the final app, the response schema of each registered service is annotated with *terms* (e.g., *title*, *description*, *address*) indicating classes of attributes, according to a vocabulary that is defined and maintained in the service repository. These annotating terms have a double role: when the mashup is defined they allow the designer to select service attributes by reasoning on abstract categories,

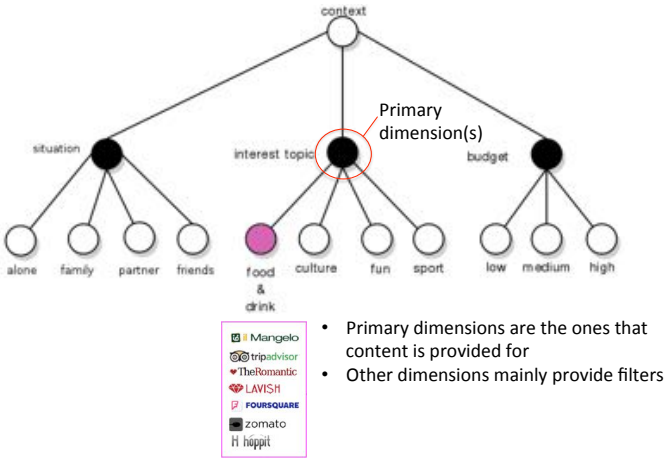


**Fig. 2.** An excerpt of a Context Dimension Tree for representing usage contexts and association of services to primary dimensions.

instead of specific attributes resulting from service queries; at run time, they facilitate merging different result sets, since it is easier to identify attributes that refer to the same entity properties.

**Universal CDT augmentation.** The administrator also specifies the *Universal CDT*. In order to support the context-aware selection of services at runtime, as reported in Figure 2 s/he also augments the context representation with *mappings* between the identified context elements and the services registered in the platform.

**Mashup Design.** The *mashup designer* starts from the image of the available resources represented by the augmented Universal CDT and, using a *Design Visual Environment*, defines a *Tailored CDT* by further refining the selection of possible contexts and the mapping with services (both core and support), to fulfill the needs and preferences of the specific users or user groups.

Given the services associated with a given context dimension (e.g., all the services providing data on restaurants associated with the `food&drink` context dimension) the designer can select the categories of attributes (i.e., the annotating terms specified at service-registration time) to be visualized in the mobile app. As schematically represented in Fig. 3, this selection is operated visually, according to a composition paradigm for mobile mashup creation already defined in [4]. The designer drags and drops the semantic terms associated with the attributes of the service response. A "virtual device" provides an immediate representation of how the final app will be shown on the client device. In addition, the designer can include in the mashup *support services* providing the user with further information. All the visual actions are translated by the design environment in a JSON-based *mashup schema*, which specifies rules that

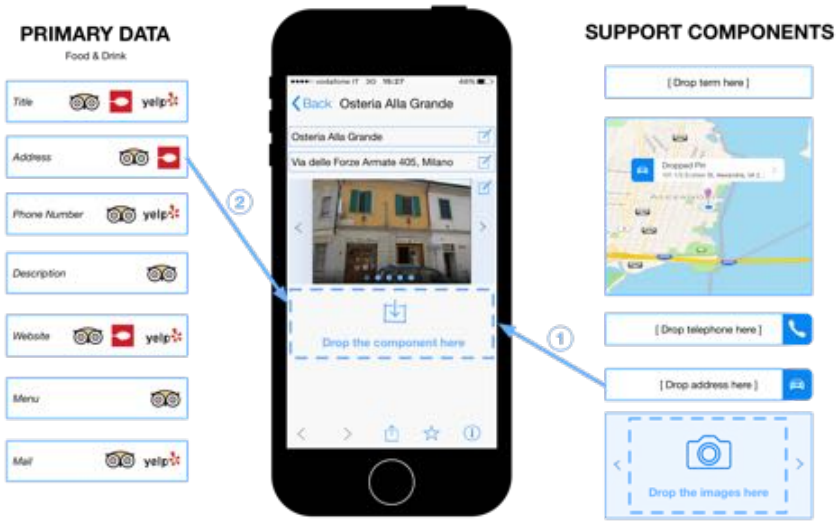


**Fig. 3.** Schematic representation of the visual mapping activities to associate service attribute classes to elements of the final app UI.

at run-time guide the instantiation of the resulting app and the creation of its views.

In addition to this, s/he can refine the association with support services, where needed, to enrich the user experience (e.g., provide transport indications to reach a restaurant, or extending the core content with description of places taken from Wikipedia). Support services are also context dependent: for instance, if the user expresses that s/he is in a situation where s/he wants to use "transportation by car", the system provides route information; otherwise, if s/he selects "public transport" it suggests a bus line.

**App Execution.** The *CAMUS (app) users* are the final recipients of the mobile app that offers a different bouquet of content and functions in each different situation of use. When the app is executed, the context elements that characterize the current situation, identified by means of a client-side *sensor wrapper*, are communicated to the server; this, in turn, chooses the correct services to be invoked and returns data in an integrated format. The mashup schema created by the designer is interpreted locally (by means of a *Schema Interpreter*) and the generated views are populated with the returned data. The platform indeed exploits generative techniques: modeling abstractions guide the design of the final applications, while generative layers mediate between high-level visual models and low-level engines that execute the final mashups. Execution engines, created as hybrid-native applications for different mobile devices, then make it possible the interpretation and pervasive execution of schemas.

## 3 Platform Architecture

The architecture of the final system is server-centric. The framework used to develop the *Server* is Node.js and the database used is MongoDB. The Server's main functionality is to provide the integrated result set to the mobile app. This process involves *i)* the analysis of the user's context to select the services to be queried and, *ii)* querying the selected services and transforming their results into an integrated data set to be rendered by the mobile app.

The Server exposes several endpoints to enable the execution of service queries. The main API is compliant with the GraphQL API specification [5], which offers a layer enforcing a set of custom-defined typing rules on the data sent and received via HTTP. Besides, it provides a flexible way to specify the response format, by making it easier to support different generations of APIs.

The *Visual Design Environment* consists of a suite of Web applications to: *i)* register new services, *ii)* specify visually (and automatically generate an internal representation of) the CDTs and the associations of services with pertinent nodes, and *iii)* design the final mashups and generate their schema.

The *Client App* manages the interaction of the end-user with the whole system. During its initialization, the app loads the user CDT and the mashup schemas to be rendered. It is implemented using React Native [6], a framework recently introduced by Facebook to streamline the production of cross-platform mobile apps. The app logic is written in Javascript and is agnostic with respect to the target platform. A typical request from the Client App is composed of a JSON payload that describes the *context* and a specification of the format of the data that is expected by the client. The request is thus processed through the following steps:

- A *Context Manager* at the server side parses the context and "decorates" it with all the Augmented UCDT information (services, ranks, etc.) related to its elements.
- Based on analyzed context, a *Service Selection* component selects the services to be queried.
- A *Query Handler* queries the selected services by using service-specific bridges that wrap the retrieved result sets and transform them into a common internal representation that complies with the semantic terms associated to the different service attributes. This internal representation enables merging the different data sets based on attributes associated with the same terms.
- Finally, possible duplicates are removed and the activation of support services - if any, is bound to the selection of specific attributes in the integrated result set, as defined by the mashup designer when creating the mashup.

Once built, the integrated data set is sent back to the app. The instantiation of views is driven by the JSON mashup schema previously downloaded and uses the React Native cross-platform technology to build the resulting user interface elements.

## 4 Feature Checklist

- **Mashup Type:** Data and UI mashups
- **Component Types:** Data Components, Logic components, UI components
- **Runtime Location:** Both Client and Server
- **Integration Logic:** Orchestrated integration
- **Instantiation Lifecycle:** Long-living

*Mashup Tool Feature Checklist:*

- **Targeted End-User:** Local Developers
- **Automation Degree:** Full Automation
- **Liveness Level:** Level 3 (Automatic Compilation and Deployment, requires Re-initialization)
- **Interaction Technique:** Visual Language (Iconic), WYSIWYG
- **Online User Community:** None.

## References


1. Corvetta, F., Matera, M., Medana, R., Quintarelli, E., Rizzo, V., Tanca, L.: Designing and developing context-aware mobile mashups: The CAMUS approach. In: Proc. of ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015. (2015) 651–654
2. Daniel, F., Matera, M.: Mashups - Concepts, Models and Architectures. Data-Centric Systems and Applications. Springer (2014)
3. Bolchini, C., Curino, C., Orsi, G., Quintarelli, E., Rossato, R., Schreiber, F.A., Tanca, L.: And what can context do for data? Commun. ACM **52**(11) (2009) 136–140
4. Cappiello, C., Matera, M., Picozzi, M.: A UI-Centric Approach for the End-User Development of Multidevice Mashups. TWEB **9**(3) (2015) 11
5. Facebook: GraphQL. Draft RFC Specification, `https://facebook.github.io/graphql` (2015)
6. Facebook: React Native. React Native official page, `https://facebook.github.io/react-native` (2015)