Defining User-Generated Services in a Semantically-Enabled Mobile Platform

Marcin Davies  
Telecommunications Research Center Vienna (FTW)  
Vienna, Austria  
davies@ftw.at

François Carrez  
University of Surrey  
Guildford, Surrey, UK  
f.carrez@surrey.ac.uk

Diego Urdiales  
Telefónica I+D  
Spain  
diegou@tid.es

Anna Fensel  
Telecommunications Research Center Vienna (FTW)  
Vienna, Austria  
fensel@ftw.at

Maribel Narganes  
Robotiker-Tecnalia  
Spain  
maribel@robotiker.es

José Danado  
YDreams  
Portugal  
jose.danado@ydreams.com

ABSTRACT
Mobile computing enables end-users not only to access and consume information on-the-go but also to act as service and content providers. With the right tools, end-users can create small services on their mobiles and share valuable and context-aware information with others.

This work introduces a platform for end-user generated mobile services — so-called microservices. As a key component we present a microservice description language for user-driven mobile service creation, powerful service discovery, and platform-independent service execution and rendering. The paper also gives insight into the visual authoring tool and the user feedback gathered in evaluations.

Categories and Subject Descriptors
H.3.4 [Information Storage and Retrieval]: Systems and Software; D.2.6 [Software Engineering]: Programming Environments—Graphical Environments

General Terms
Design, Languages

Keywords
Semantic Web Services, Microservices, Mobile services, End-user Development, User-centric Service Creation

1. INTRODUCTION
This paper presents a new paradigm for communication between mobile users, relying on a novel service infrastructure that enables end-users to become main actors in the process of service generation and provisioning. The potential of mobile users as service providers is high since the mobile, as a user-attached device, provides an always-on open door to the world. Myriads of knowledge\(^1\) generated by millions of potential users can be made available through this kind of services. Moreover, services are not only accessible from the mobile but they can be also created “at the point of inspiration” from the mobile itself. This new kind of services will allow users to obtain and provide semantically enriched content, like opinions and recommendations, to fellow users. With this purpose, our work defines a service architecture, a set of mobile tools and a platform to allow mobile users to create small, sharply focused and knowledge-based mobile services, so-called microservices.

Some examples of usage scenarios for microservices are:

- **Traffic Jam Killer**: Users publish their location, time and speed at given intervals during their car trip and information from all users is aggregated in order to assess the current traffic situation at a given time;
- **Translator**: A linguistic expert user decides to share his knowledge and creates a microservice that is able to receive small pieces of text from other users and to provide its translation in real time.

The main objectives of this paper are to explain the infrastructure components and in particular the microservice description language as a key enabler for user-generated services. Consequently, this paper is structured as follows: In Sect. 2, we provide an overview of the general architecture, the language for end-user driven microservice creation and its role within the service framework. The semantic aspects of the microservice description are discussed in Sect. 3 and the non-semantic parts are outlined in Sect. 4. Sect. 5 gives insight into the visual authoring tools and Sect. 6 discusses the user feedback gathered in early evaluations. Related work is considered in Sect. 7. Sect. 8 concludes the paper and gives an outlook on future work.

2. OVERVIEW OF THE MICROSERVICE PLATFORM
\(^1\)In this work, we represent knowledge as semantically annotated data and content
As explained in the introduction, microservices are small and sharply focused services, which are designed by end-users on their mobile phones. Microservices can be used to share pieces of context-based content (opinions, recommendations, pictures, audio, etc.) that is collected either automatically, by terminal capabilities, or manually, by direct user input. This section introduces the overall platform and the microservice description language.

2.1 Architecture

The microservice architecture is composed of the following key components:

- The Service Creation Kit (SCK), a mobile application used to create new microservices from scratch or to customize existing microservices;
- The Service Execution Environment (SEE), which executes the microservices on the mobile device;
- The Service Warehouse (SWH), which is the network-side component where microservices are bundled and stored; and, finally
- The Knowledge Warehouse (KWH), which is the network-side registry where semantic descriptions of microservices and their contents are available.

Whereas the SCK and SEE are terminal-side components, the SWH and KWH are located in a central place in the network side. The exchange of content is peer-to-peer (there is no knowledge or content stored in the network), but the architecture is centralized as far as the repository (SWH) and registry (KWH) are concerned (this is a design choice — and its role within the system architecture).

At the terminal side, the SCK works with so-called Simple Service Elements (SSE’s) as building blocks that provides the user with an abstraction of the underlying language and capabilities (cf. Sect. 3.3 and 4). Those SSE’s are then mapped to capabilities, residing either in the terminal or in the network, which are invoked at execution time by the SEE.

At the network side, the search engine is the main component linking the SWH and the KWH. It accepts search requests for:

- microservices and related content relevant for a user in a given context;
- microservices that users can modify to make their own services;
- Simple Service Elements (SSE’s): service building blocks that can be used for service creation.

The general (and simplified) microservice lifecycle is depicted in Fig. 1 (see also [7]). At a first stage, users will create services by combining the previously mentioned SSE’s or by customizing an existing service. Microservices will then have to be deployed in order to make them searchable and executable in the embedded SEE of the users’ terminals (it should be noted that we distinguish between providers of a service and simple users, i.e. consumers that just access a microservice).

2.2 UDL: Microservice Description Language

As already motivated in the previous section, a description language for microservices is a central contribution of our work. This section gives an overview on the UDL — the U+ Service Description Language — and its role within the system architecture.

Microservices are interpreted by an execution environment running in the mobile terminal. Therefore, they need to be specified in a language that is:

- easily interpretable in a mobile terminal, and with direct access to terminal capabilities;
- easy to create, i.e., which can be mapped to higher level building blocks understandable by the user that creates the service;
- highly expressive, so that a wide variety of microservices can be specified using the language.

Semantic descriptions play an important role in achieving these objectives. They are needed to characterize what the microservice is about and what kind of content it provides. Moreover, they enable much more precise microservice discovery by other members of the community, and an easier and more accurate microservice creation process by annotating service building blocks.

2.2.1 Parts of the UDL

The UDL is split into five sub-languages, which are briefly introduced in the following and illustrated in Fig. 2.

- **Microservice Profile:** The UDL Service Profile (UDL-SP) provides a descriptive and functional definition of the service (i.e. “what the service is doing”). Essentially, the UDL-SP is an ontology with a set of metadata that are ultimately used by the search engine in order to discover microservices that match specific semantic properties (similar to the Service Profile ontology of OWL-S [1]). These properties can be expressed either with regard to the microservice ontology (e.g. “find all services of type blog”) or a content ontology (e.g. “find all services dealing with a particular kind of content or find any microservice created by James”).

This content ontology (UDL-CD) is explained shortly below and in more detail in Sect. 3.2. More information on the Microservice Profile is given in Sect. 3.1.

- **Microservice Content Description:** The UDL Content Description (UDL-CD) considers the definition and the properties of service content (content is what any microservice is ultimately producing or sharing). Content descriptions are also ontology based and rely on existing domain ontologies. They are discussed in more detail in Sect. 3.2.

\(^{2}\) U+ Service is a project-internal term for microservices.
Microservice Capability Profile: The UDL Capability Profile (UDL-CP) provides semantic descriptions of SSE’s (in an ontology) for ensuring a quicker and more accurate creation process as only blocks that can be linked (in terms of IOPE’s) to existing blocks would be proposed to the user in the SCK. The UDL-CP is explained in detail in Sect. 3.3.

Microservice Logic: The UDL Service Logic (UDL-SL) gives an operational definition of the microservice. A formal definition of the elementary operations within a microservice — the previously mentioned SSE’s — is necessary for service execution (by the SEE). The execution of a microservice results in either

- the creation/modification of content by the microservice provider or
- the access and/or modification of content by a microservice user or
- exchange of content between microservice users.

With regard to the service creation paradigm the project pursues a block-based approach, where users are able to specify the functionality of a service (cf. Sect. 4). In any case, running a microservice results in the execution of elementary steps which have to be defined precisely in the UDL-SL (hence, this task is close to the traditional definition of a programming language). The UDL-SL definition is an XML-based language, which borrows from the WS-BPEL [5] specifications. More information can be found in Sect. 4.1.

Microservice Rendering: The UDL Service Rendering (UDL-SR) considers the rendering of microservices on a mobile terminal. Because mobile terminals can vary in nature and characteristics an abstraction layer is needed that describes in a generic way how each elementary piece of a microservice is rendered on the user’s mobile terminal. Therefore, the UDL-SR language proposes ways to define all rendering objects (i.e. user interface elements) and aspects associated with the service logic at execution time. The UDL-SR is discussed in detail in Sect. 4.2.

In conclusion, these five sub-languages/ontologies are used to describe microservices and the SSE’s they are made of. A number of templates and ontologies are used at various places in the (simplified) architecture, which is shown in Fig. 3. The purpose of the next section is to explain how and where the individual parts of the UDL are used.

2.2.2 UDL within the Architecture

The UDL itself is not part of the architecture as such, but the templates and ontologies which result from a microservice, its content and all SSE descriptions have to be stored and used within the architecture. The UDL-SL (describing service logic) and UDL-SR (for the service presentation or rendering) templates are produced by the SCK and run by the SEE. It is worth noting that reverse engineering is enabled, which means that it is possible to re-build the original graphical view from the UDL-SR and UDL-SL templates within the SCK (cf. Sect. 5). This property is essential to guarantee that a user can download an existing microservice, load it into the SCK and customise it, thus reducing the efforts needed for service creation as more microservices become available in the framework.

The service profile ontology used for the UDL-SP describes what the microservice does. All metadata is expressed with regard to this ontology. Metadata elements are used and stored at various places within the architecture (cf. Fig. 3):

- The SEE works with metadata elements that influence the way a microservice is executed locally, i.e. operational metadata (subset of UDL-SP translated in XML);
- The SWH processes metadata that influence the way
the service is published, accessed, and viewed by others (subset of UDL-SP translated into XML); 

- The KWH stores and uses the metadata elements that influence the way a service can be discovered (the whole UDL-SP in its native form, i.e. the form resulting from the SCK, which can also be called "internal form").

Storing and managing microservice descriptions is the main function of the SWH and KWH, which can be seen as an analogy to the concepts of Repository and Registry within SOA, respectively (cf. [13]). The role of the SWH is the Repository (i.e. the place where microservices are stored and from/to where they can be downloaded/uploaded), whereas the KWH plays the role of the Registry (i.e. the place where microservices are published and described, and where lookup functions take place).

The semantic descriptions of the SSE's (UDL-CP, Sect. 3.3) and of service-specific content (UDL-CD, cf. Sect. 3.2) are also stored in the KWH and can be accessed by the search engine. While the service profile and the content description are of primary interest for end-user search requests, SSE descriptions are actually accessed by the SCK during the service creation process (cf. Sect. 3.3).

3. MICROSERVICE SEMANTIC DESCRIPTIONS

As introduced in Sect. 2.2, the semantic description of a microservice consists of three main parts: (1) UDL-SP: Service Profile, (2) UDL-CD: Content Description, and (3) UDL-CP: Capability Profile.

3.1 UDL-SP: Microservice Profile

The service profile provides a descriptive and functional definition of the service as a whole entity. It contains a series of metadata elements that are described through ontologies. Attributes are applicable to each of the metadata elements to define aspects such as access restrictions, whether a metadata element is mandatory or optional, searchable or not, etc.

Every Microservice Profile consists of

- **Global Metadata**: This common set of metadata fields (e.g. service name, description, etc.) is pre-defined and their attribute values are fixed. These elements are mainly used by the search engine;

- **Specific Metadata**: This optional set of specific metadata can be added and defined by the service creator if needed.

It should be noted that because of their different purpose (e.g. operational, local, searchable — see Fig. 2) metadata elements are stored in different parts of the architecture (cf. Fig. 3). Listing 1 shows a sample from a typical microservice profile.

**Listing 1: Microservice Profile Example**

```xml
<ServiceProfile>
  <GlobalMetadata>
  <Name>Friend Locator</Name>
  <Description>Sharing your Location with your Friends</Description>
</GlobalMetadata>
</ServiceProfile>
```

3.2 UDL-CD: Microservice Content Description

This part of the UDL describes the content that a microservice produces and receives by mapping the service content items to a commonly agreed content ontology. The content structure is also used within the service logic (cf. Sect. 4.1). Depending on the service creation paradigm used, this representation can be explicit (mapping content to UDL-IL variables) or implicit (by using the result of a block as input for a slot within the content structure). Listing 3.2 shows a short snippet for describing Times Square.

**Listing 2: Microservice Content Description Example**

```
<Name>Friend Locator</Name>
```

3.3 UDL-CP: Microservice Capability Profile

As already mentioned before, microservices are generally built out of SSE’s. In order to allow a quick, user-friendly, and meaningful service creation (e.g. by recommending only suitable and matching service components) these SSE’s are semantically described. As the SSE’s are different in nature, the way they are described may vary:

- **Embedded Capabilities**: These capabilities are part of the execution environment installed in the mobile terminal, and are therefore available to all terminals. Even if they might be specialised according to the terminal maker and type, they should feature the same interface;

- **External Capabilities**: This kind of capabilities are linked to functionalities that reside within the network. They are available through the SWH and are discovered by searchable metadata (stored at the KWH). These ‘external’ components can be downloaded and installed inside the mobile terminal. After they have been installed, they can be used and combined like any other embedded capability;

- **External Terminal Capabilities**: These are additional capabilities that are provided by terminal makers and telecommunication operators. They are designed to...
work within a terminal, exploiting their embedded devices;

- **General purpose blocks**: These blocks are not terminal dependent and could be provided by any 3rd party;
- **Information/content blocks**: These represent pieces of content that exist as such and do not result from the call of a capability.

All of these “objects” exist in the user environment and are manipulated by a user within the SCK. Any resulting microservice will be a combination of such objects with a “touch of control” (conditionals, loops etc. – cf. Sect. 4.1). Listing 3 shows the description of a calendar capability for illustration.

**Listing 3: Microservice Capability Example**

```xml
<EmbeddedCapability rdf:ID="CalendarCapability">
  <implementsConcept rdf:resource="#RescheduleEvent"/>
  <implementsConcept>
    <FunctionalConcept rdf:ID="AddCalendarEvent">
      <inputs rdf:resource="#AddCalendarEvent"/>
      <takeInputs rdf:resource="#AddCalendarEvent"/>
    </FunctionalConcept>
  </implementsConcept>
  <implementsConcept>
    <FunctionalConcept rdf:ID="GetCalendarEvents">
      <outputs rdf:resource="#GetCalendarEventsList"/>
      <takeInputs rdf:resource="#GetSearchTerm"/>
    </FunctionalConcept>
  </implementsConcept>
</EmbeddedCapability>
```

### 4. MICROSERVICE OPERATIONAL AND PRESENTATION DESCRIPTIONS

As outlined in Sect. 2.2, the operational and presentation descriptions of a microservice are divided into (1) **UDL-SL**: Service Logic (operational description) and (2) **UDL-SR**: Service Rendering (presentation description).

UDL-SL and UDL-SR are domain-specific XML-based description languages to define the logical workflow and visual layout of a microservice, respectively. The proposed paradigm mimics a tree structure using a top-down path. Whether for logic or visual layout, a graphical representation can be achieved by representing nodes within the tree as visual blocks within the SCK interface (which will be explained in more detail in Sect. 5). Figure 4 provides an overview on how the information flows within a block-based approach.

#### 4.1 UDL-SL: Microservice Logic

The Microservice Logic defines the functional aspects of a service seeking for high expressiveness and high usability for end-users. However, since high expressiveness unfortunately often results in high complexity, UDL-SL seeks for a trade-off between both in order to make microservice creation attractive for end-users. Although UDL-SL implements common programming constructs (loops, conditionals, etc.), it focuses on a relatively simple tree structure that also provides the needed visual representation.

Listing 4 shows a small example where a basic data structure and two procedures for sending/receiving are defined.

**Listing 4: Microservice Logic Example**

```xml
<logic>
  <variables>
    <structure name="MsgTemplate">
      <items>
        <field name="content" type="text" />
        <field name="location" type="location" />
      </items>
    </structure>
    <variable name="message" type="MsgTemplate" />
  </variables>
  <procedures>
    <procedure name="gotRevMessage">
      <do>
        <invoke name="addContent" capability="Internal" out="void">
          <parameters>
            <parameter value="Table_Message"/>
            <parameter value="message"/>
          </parameters>
        </invoke>
      </do>
    </procedure>
    <procedure name="sendAMessage">
      <do>
        <send ID="1" recipients="Users" type="MsgTemplate" data="myMessage"/>
      </do>
    </procedure>
  </procedures>
</logic>
```

#### 4.2 UDL-SR: Microservice Rendering

The Microservice Rendering defines graphical elements for a microservice and addresses mobile device peculiarities such as varying screen size, limited interaction modalities, and different operating systems. Therefore, UDL-SR defines an abstract set of UI elements to achieve a platform and device independent design.

Similar to UDL-SL, UI elements defined within UDL-SR are represented through a block-based approach. For instance, UDL-SR defines UI elements such as: “presentation”,...
“window”, and “panel”. These elements are represented in a tree with presentation as the root node and windows as child nodes. Each window also has a child node, namely a panel that can contain additional UI elements. This kind of hierarchy defines the way the UI is rendered. Listing 5 shows how a DataGrid is defined in UDL-SR as an example.

Listing 5: Microservice Presentation Example

```xml
<DataGrid ID="myDataGrid">
  <Items>
    <DataGridRow>
      <Items>
        <DataGridColumn value="a"/>
        <DataGridColumn value="b"/>
      </Items>
    </DataGridRow>
    <DataGridRow>
      <Items>
        <DataGridColumn value="c"/>
        <DataGridColumn value="d"/>
      </Items>
    </DataGridRow>
  </Items>
</DataGrid>
```

5. MOBILE AUTHORING TOOL

As Sect. 4 mentioned, UDL follows a tree structure. Thus, using XML as the basis was an obvious choice that has the advantage of being directly representable as visual blocks in the user’s terminal. This section shows the tools used for visualising the UDL language components in the mobile terminal.

Being an XML-based language the UDL can easily be represented in a tree structure on a mobile terminal. However, display size restrictions on the one hand and information overload on the other hand make a complete visualization senseless. Therefore, different levels of visualization and complexity are possible and the user can easily change between the different views of the UDL language (“gentle slope” approach, cf. [12]). The first view aimed at “beginners” is a wizard-like approach asking the user different questions, i.e. the type of the service, the name, etc (cf. Fig. 5). The second, intermediate view is similar to the beginner view, but offers more customization possibilities. The third view targets experienced users with a visual UDL editor (cf. Fig. 6).

The advanced editing mode allows the mobile user to manipulate different aspects of the UDL, i.e. service profile, content, logic, and presentation in so-called “contexts”. Each context provides a “world view” where users can seamlessly transverse the nodes of the blocks in that context. The user is able to add, modify or remove blocks. Recommendations to the user (which are currently based on the inputs a parent node accepts from child nodes) should help in selecting a fitting block.

Figure 6 shows screenshots of the context selection, the world view of the presentation, recommendations of fitting blocks, and a small snippet of UDL that is generated from the visual editor, i.e the SCK. The UDL acts as an intermediate language — services defined in UDL can be shared among other users and customized or re-edited through the SCK. For direct execution on a mobile platform UDL is converted into xHTML + Javascript (so it can run in a mobile web browser).

6. EVALUATION RESULTS

In order to get early feedback on the authoring tool and the expectations and background of users, we conducted a small user trial and used an online questionnaire.

The trial was done with five users (which were rather technologically inclined) performing six different tasks necessary for microservice creation. In general, the test showed that users with a technological background can understand the concepts behind the creation of a microservice. For the beginner view the participants recommended usage of navigation tips and less technical jargon to make the process easier. Although the basic concepts of the advanced view were easily perceived, users found it still too complex to use. The tests showed that the interface could be improved through introduction of help tips, short descriptive text, icons, color codes and templates.

The online survey contained 38 questions and tried to assess both views of the authoring tool (besides gathering demographic data and information about background and experiences with similar tools, which will be not discussed in detail here). We received a total number of 52 full responses over a period of 4 months. 74 % of the respondents were male and 98 % from Europe. 88 % had a university degree (bachelor or higher), and 70 % were working in ICT or science fields. 44 % of the participants were between 20-29, one third between 30-39, and the remainder (24 %) between 40-60 years old. 46 % of the respondents confirmed to access the internet from their mobiles. Two thirds of the respondents use the internet at least 4 hours per day.
The purposes to create services were mostly to exchange information (71 %), stay in contact (6 %), just for fun (60 %) — cf. Fig. 7. People would mostly create them while waiting (73 %), commuting (61 %) or at home (60 %). 44 % would also create services at work, i.e. professionally. Free of charge provision/usage was the most frequent answer (73 % and 87 % respectively). About a quarter of people could image to either make small payments or subscriptions for services.

During the survey, users were presented a demo of the question-based SCK and two videos of both the question-based and the block-based service creation approaches. In terms of easiness, the question-based approach was received much better (“It seems to be also suitable for beginners to create services”, “The user is well guided by questions”) than the block-based, advanced approach (“for freaks/nerds mainly”, 13 users regarded this method as too complex). The block-based approach was mainly regarded as a professional tool for creating more complex services, and one which might fit better on a desktop than on a mobile. Six users, however, mentioned that the question-based approach might be still too restricting (“I could imagine creating a service this way, but it would probably be too simple to do anything I really wanted.”). This feedback strengthened our design choice to offer different complexity levels (see Sect. 5) in a “gentle slope approach” (cf. [12]). This way end-users can start interacting with the system at a very basic level and during time progress towards becoming a “power-user”.

7. RELATED WORK AND ADVANCE BEYOND THE STATE OF THE ART

The work presented here advances the state of the art in three main areas, namely End-User Programming and Service Creation (1), Semantic Web Services (2), and User-Generated Content (3).

1. End-User Programming and Service Creation

Current examples for user-created software on the Web include mashup tools like Yahoo! Pipes\(^4\) or Microsoft Popfly\(^5\). In the mobile world, widget platforms such as Nokia WidSets\(^6\) have emerged that, at a first glance, could be seen similar to our approach. However, these offerings do not provide a platform for construction of real services (and are limited to content aggregation not provision), and can be seen complementary to our work as new microservices could be developed on top of them.

The innovations of our solution compared to existing platforms and enablers (cf. [14, 15, 16, 9]), end-user development techniques (cf. [10, 11]), and mobile service composition approaches (cf. [17]) are:

- A platform-independent and ontology-based service description language for mobile devices;
- A mobile execution environment for user-generated services;
- New end-user oriented service creation methods that run on the mobile.

2. Semantic Web Services

Semantic Web Services (SWS) are usually based on rather heavy-weight service models such as OWL-S [1] or WSMO [2]. Although more lightweight approaches such as SAWSDL [3] and SA-REST [4] exist, they are not suited for mobile services. Recent efforts towards modelling of mobile services (e.g. [19, 20]) may not be widely accepted or standardized yet. Moreover, existing toolkits (cf. [6]) for SWS are not oriented towards end-users due to their complex nature. Therefore, our work offers a description language for mobile user-generated services and user-friendly ways for mobile service annotation while on the go.

3. User-Generated Content

The presented framework builds on top of the widespread Web 2.0 trend of User-Generated Content (UGC) and extends the concept by allowing users to create their own services. Microservices are a quantum leap over UGC since they

- are dynamic, changing their output over time and letting users move beyond static content into functionality;
- give users full control over their creations, since they are provided from their own mobile terminals; and
- allow for multiple authoring, since a microservice can have several providers (see [8] for more details).

However, microservices respond to the same ultimate user needs as UGC: authorship, keeping in contact, self-broadcasting; and they have an equivalent viral potential, creating a service metropolis which is a source of relevant, on-the-go information for other users.

8. FUTURE WORK AND CONCLUSIONS

In this paper, we have presented a novel design for user-generated mobile microservices. An extensible model for the definition of a microservice and corresponding mobile authoring tools have been developed. This sets a step towards the vision of user-generated mobile microservices and their deployment in an open service platform.

---

\(^4\)http://pipes.yahoo.com
\(^5\)http://www.popfly.com
\(^6\)http://www.widsets.com
The development of a full-featured prototype infrastructure is ongoing and technical aspects, such as applicability of the derived solution to different mobile platforms and its execution on the users’ mobile devices, are being investigated. Functionalities for facilitation of efficient service creation, discovery of microservice elements, and associated knowledge management will be derived. Such advanced techniques would evolve the current “shallow” Web 2.0 of content to the future “semantically structured” internet of services. The technical characteristics of the resulting system such as its performance in a large-scale evolving Internet environment are to be evaluated as well.

One of the most important goals for the future, however, will be a thorough assessment of the prototype with a larger set of users in field trials. Finally, the aspects of end-users’ perception and acceptance of such a framework for the generation of microservices as well as the usage of semantic social software and community-driven practices (cf. [18]) are of high interest as well.

Acknowledgements
This work has been supported by the European Commission, ICT FP7 Collaborative Project m:Ciudad — A Metropolis of Ubiquitous Services.

9. REFERENCES


http://www.mciudad-fp7.org