SADIO Electronic Journal of Informatics and Operations Research

http://www.sadio.org.ar/ejs/

vol. 14, no. 1, pp. 3-21 (2015)

An energy-saving model for service-oriented mobile application development

Ignacio Lizarralde (ISISTAN - CONICET), Cristian Mateos (ISISTAN - CONICET), Alejandro Zunino (ISISTAN - CONICET)

ISISTAN Research Institute. UNICEN University. Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina. Tel.: +54 (249) 4439682 and Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Keywords: Smartphones, SOC, Web Services, Energy save, Android

Abstract. The development of mobile applications that combine Web Services from different providers --also referred as mashup applications-- is growing as a consequence of the ubiquity of bandwidth connections and the increasing number of available Web Services. In this context, providing higher maintainability to Web Service applications is a worth of matter, because of the dynamic nature of the Web. EasySOC solves this problem by decoupling mashups from application components. However, mobile devices have energy constraints because of the limitations in the current battery capacities. This work proposes a model that builds on the benefits of the EasySOC approach and improves this latter by assisting developers to select Web Service combinations that reduce energy consumption. We evaluated the feasibility of the model through a case study in which we compare the estimations provided by the model against real energy measurements and two handsets. The results indicated that our model had an efficacy of 81-85% for the analyzed case study.

1 Introduction

The number of mobile devices is nearly the world population (7.3 billion). As a consequence, mobile data traffic last year was nearly 18 exabytes. These facts are a conse-

quence of the growing popularity of mobile devices [22,20]. They evolved from offering limited functional capabilities such as calendars or calculators to providing the same functionality as a small computer [21]. Although these devices have enough power to be used in distributed environments [22], in general battery-life is negatively affected when resource usage grows because battery capacity has not increased accordingly to the energy consumption of recent mobile hardware [10]. Thus, while 10 years ago battery duration was about one week, today is at most few days [10].

On the other hand, nowadays, mobile devices make intensive use of networking interfaces such as Wi-Fi, 3G and HDSPA for accessing data and resources on the Internet [2,23]. However, as mentioned earlier, battery-life is a main concern in these devices, and networking interfaces highly affects energy consumption [1,15,17]. Thus, these concerns enforce the importance of energy consumption reduction in mobile devices.

Recently, Service-oriented Computing (SOC) [6] started to become popular due to the ubiquity of high speed network connections. SOC is a computing paradigm that promotes the composition of external software components, called services, available and invokable through the Internet. In this context, applications are developed using services as the basic building blocks, decreasing the cost and maintainability of the development process. Additionally, development of SOC applications that combine services in mobile devices has been made attractive for developers because of the widespread availability of Web Services --the most common technological materialization of SOC-- and the popularity of social networks (i.e., Facebook, Twitter, etc) and cloud services. These applications, which rely on a combination of Web Services from different sources, are called mashup applications. For example, SongDNA allows users to search for individual songs based on data collected from over 10 sources and obtain song lyrics, artist bio, links to videos and even information about what people are saying about the song in Twitter.

Broadly, despite their benefits, employing Web Services to develop mobile applications has some shortcomings because Web Service invocation is a resource demanding process. Moreover, these devices have limited battery-life and processing capabilities. In this paper, we propose an energy-aware SOC development model, which builds on the benefits of EasySOC [7], an approach to develop SOC applications that focus on maintainability by decoupling client application components from service interfaces. EasySOC isolates the user-provided (or in-house) application components at the client side from the interface(s) exposed by the provider(s) by means of service adapters which transform the provider interface into the expected client interface. Moreover, our model improves EasySOC by reducing energy consumption due to interface adaptations from applications by allowing users to select the least energy expensive service combinations. In other words, our model focuses on adapter energy consumption. Additionally, because of the popularity and wide spread adoption of the Android platform, this work focuses on Android-powered devices.

The rest of this paper is organized as follows. First, in Section 2 we describe EasySOC and present the proposed estimation model. Then, in Section 3 we describe the experimental evaluations performed to assess the model. Finally, in Section 4 and Section 5 we describe the most relevant related work and the conclusions, respectively.

2 An energy-saving estimation model for EasySOC

SOC promotes the reuse of software components available from the Internet, called services. Moreover, Web Services materialize SOC by prescribing languages to specify a public interface to describe the functions --operations-- and parameters required to invoke a service. Then, if a developer wants to build a mashup application he/she needs to select which available services will meet their functional requirements by inspecting their public interfaces. This ties the client application components with the interfaces of the selected services. The approach to SOC development described in [7], EasySOC, solves this problem by decoupling mashups from services and hence providing higher maintainability levels to SOC applications. The approach combines DI (Dependency Injection) and the Adapter pattern to isolate the user-provided application components at the client side from the interface(s) exposed by the provider(s). In this context, the approach requires to firstly define an expected interface for external services at design time. Then, services adapters carry the responsibility for transforming these interfaces into the interfaces exposed by the available service providers.

The model we will present builds on the benefits of the EasySOC approach, but at the same time improves this latter by providing energy estimations to EasySOC applications. This allows users to select the least energy expensive service combination (due to interface adaptations), thus reducing the wasted energy during the invocation process and hence mashup usage. However, this work does not provide a mechanism for service selection, instead, it provides a model that will help developers in service selection by providing energy consumption estimations.

Section 2.1 describes EasySOC [7,13]. Later, in Section 2.2 we will introduce the concept of micro-operation and how it relates to the model. Finally, Section 2.3 describes the model.

2.1 EasySOC

Although SOC simplifies application development through component composition, application maintainability may be affected. SOC applications rely on services that are invoked through an interface exposed by a service provider. However, service interfaces could change affecting client applications, since some of their components need to be adapted in order to meet the new interfaces. In this section we describe EasySOC [7,13], an approach that aims to isolate client application components from service interfaces trough service adapters. The energy consumption of these service adapters is the focus of the model we will present in the following sections.

EasySOC builds on the idea that an interesting implication of using DI in SOC is that client-side application logic can be isolated from the details for invoking services (e.g., URLs, namespaces, port names, protocols, etc.). With this in mind, a developer thinks of a Web Service as any other regular component providing a clear interface to its operations. If a developer wants to call an external Web Service S with interface I_S from within an in-house component C, a dependency between C and S is established

through I_S. This kind of dependency is commonly managed by a DI container that injects a proxy to S (let us say P_S) into C. At runtime, the code of C will end up calling any of the methods declared in I_S through P_S, which transparently invokes the remote service. Interestingly, this mechanism is not intrusive, since it only requires to associate a configuration file with the client application, which is used by the DI container to determine which components should be injected into other ones.

Although DI provides a fitting alternative to cleanly incorporate a Web Service into a client application, it leads to a form of coupling through which the application is tied to the invoked service interfaces (i.e., $\boldsymbol{I_S}$). In this way, changing the provider for a service requires to adapt the client application to follow the new interface. This is illustrated from an architectural perspective in the left side of **Fig. 1**, in which the layer containing the business logic depends on particular proxies (grey layers are coupled to the underlying ones). At the implementation level, this means to rewrite the portions of the application that depend on $\boldsymbol{I_S}$, which includes operation signatures that are likely to differ from that of the new interface.

To overcome this problem, EasySOC combines the DI and Adapter patterns to introduce an intermediate layer that allows developers to seamlessly shift between different interfaces. Conceptually, instead of directly injecting a layer of service proxies ($\boldsymbol{P_S}$) into the application, which requires modifying the layer containing the client business logic to make it compatible with the service interfaces ($\boldsymbol{I_S}$), EasySOC injects a layer of service adapters (see the right side of **Fig. 1**). To evidence the benefits of applying this approach, suppose a client application that uses a weather service to obtain information about the temperature. If the service provider shuts down the service permanently, it is necessary to change the service provider. This means that the client application components invoking the old service must adapt to the new service interface. On the other hand, for an EasySOC application only the adapter of the old service needs to be changed, leaving intact the application components.

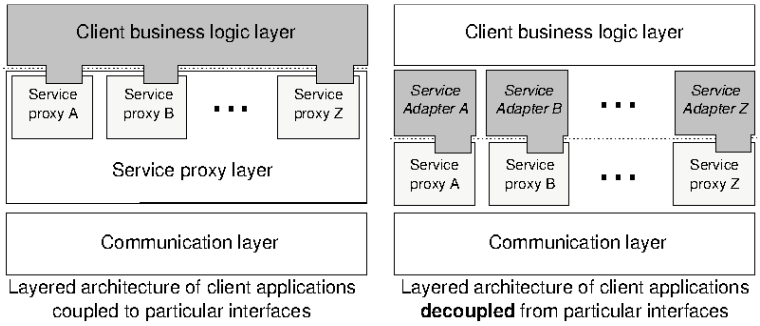

Fig. 1. Not using (left) and using (right) EasySOC in client applications: Architectural differences.

A service adapter is a specialized Web Service proxy, based on the Adapter pattern, which adapts the interface of a particular service according to the interface (specified by the developer at design time) expected by the in-house components. We refer to A_{sc}

as an adapter that accommodates the actual interface of a service S to the interface expected by an in-house component C. In other words, an adapter carries the necessary logic to transform the operation signatures of the expected client interface to the actual interface of a selected Web Service. For instance, if a service operation returns a list of integers, but the application expects an array of floats, a service adapter would perform the type conversion.

2.2 Micro-operations

In the previous section we have described EasySOC, which isolates the client logic from the service interface exposed by providers. In this context, each adapter performs a set of operations (from now on referred as micro-operations) to transform the signature of the expected client interface to the interface expected by the provider.

As mentioned earlier, this work aims to create a model for estimating the energy consumption of Web Service interface adaptations. To achieve this, we followed three steps.

First, we determined which micro-operations are involved in the adapters commonly built. Second, we measured the energy consumption of each micro-operation. Third, we applied the model explained in Section 2.3 to estimate the overall energy consumption. This section covers the first step.

Firstly, we created a catalog containing the most common operations involved in interface adaptations. Micro-operations were selected as a subset of the Java Grande6 benchmark suite, similarly to the studies presented in [20]. Table 1 shows this catalog along with the Java equivalent code of each micro-operation. There are two groups of microoperations, one group containing primitive operations (i.e., cast int to float, cast int to double) and another group involving complex operations (i.e., create an Object, create an int array). In order to show the applicability of micro-operations for describing service adapters, let us suppose a reverse-geocoding service:

– Client (expected interface): String getAddress(int latitude, int longitude).
– Provider (real interface): String getAddress(GPSMessage_Type2 message).

Fig. 2. A complex type of the expected interface and Fig. 3. Example: Adapter Implementation show the structure of the GPSMessage_Type2 complex type and the Java code of the adapter responsible for transforming the client expected interface into the actual service interface, respectively. The resulting description of the necessary adaptation in terms of micro-operations is as follows:

Table 1. Micro-operations catalog

Micro-operation name	Equivalent Java code

MOP1: cast int to double	`int a;`
	`double b;`
	`b = (double) a; o a = (int) b;`

MOP2: cast int to float	`int a;`
	`float b;`
	`b = (float) a; o a = (int) b;`

MOP3: cast long to double	`long a;`
	`double b;`
	`b = (double) a; o a = (long) b;`

MOP4: cast long to float	`long a;`
	`float b;`
	`b = (float) a; o a = (long) b;`

MOP5: create an Object instance	`Object a = new Object();`

MOP6: assign from other instance	`public class A {`
	`  public int a;`
	`  public int b;`
	`}`
	`public class B {`
	`  public void foo ( ){`
	`  A aInstance = new A( );`
	`  a Instance.a = aInstance.b;`
	`  }`
	`}`

MOP7: assign from the same instance	`public class A{`
	`  private int a;`
	`  private int b;`
	`  public foo (){`
	`  a = b ;`
	`  }`
	`}`

MOP8: create a float array[n]	`float [n] a = new float [n];`
MOP9: create a int array[n]	`int [n] a = new int [n];`
MOP10: create a long array[n]	`long [n] a = new long [n];`
MOP11: create a Object array[n]	`Object [n] a = new Object [n];`

- 1 occurrence of MOP5: GPSMessageType2, line 11.
- 2 occurrences of MOP1: (double) longitude and (double) latitude, lines 12 and 13.
- 2 occurrences of MOP6: setLongitude(longitude) and setLatitude(latitude), lines 12 and 13.

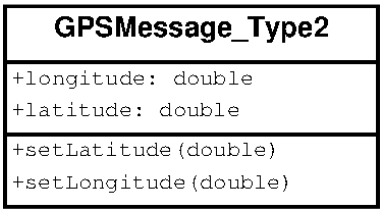

Fig. 2. A complex type of the expected interface

```
1. public class GetAddress_V2_MultipleAdapter implements IGe-
   tAddressAdapter {
2.   private ServiceMultipleConsumerProxy proxy;
3.   public GetAddress_V2_MultipleAdapter  (ServiceMultipleCon-
   sumerProxy proxy) {
```

```
4.      this.proxy = proxy;
5.    }
6.    public String getAddress (int latitude, int longitude) {
7.      GPSMessage_Type2 msg = new GPSMessage_Type2 ( );
8.      msg.setLatitude((double) latitude);
9.      msg.setLongitude ((double) longitude);
10.     return proxy.getAddress (msg);
11.   }
12. }
```

Fig. 3. Example: Adapter Implementation

To conclude, in this section we have presented a catalog of micro-operations for describing data transformations and how to use this to represent an adapter in terms of such elemental adaptation operations. The estimation model covered in the next section uses these descriptions along with the energy consumption of each micro-operation as an input to estimate the overall energy consumption of an adapter. Also, we will expand this idea to estimate the energy consumption of mashups.

2.3 Estimation model

The model proposed in this section helps to estimate the least energy expensive combination of services when building a mashup. In order to achieve this, the model relies in the concept of micro-operation we have described earlier.

Formally, a Web Service mashup is a combination of one or more services from different sources $\{S_1..S_n\}$. The EasySOC approach establishes that every service has associated an adapter that carries the necessary logic to transform the signatures of expected interfaces into the actual service interfaces. Formally, for each service $\{S_1..S_n\}$ there is an adapter $\{A_1..A_n\}$ each one having an energy cost, namely $\{C_1..C_n\}$. Additionally, an adapter can be described in terms of micro-operations $\{MO_1..MO_p\}$. Then, the estimated energy cost of a service adapter can be calculated as $\mathrm{C_k} = \sum_{i=1}^{p} \mathrm{CMO_i}$, where CMO_i is the energy cost of the corresponding MO_i micro-operation. Finally, the overall estimated energy cost of a mashup k (M_k) is $\mathrm{CME_k} = \sum_{i=1}^{n} \mathrm{C_i}$.

To illustrate the model, suppose a developer who wants to select the least energy expensive combination of services to get information about songs (e.g., SongDNA). The developer wants to get links to the song videos, and the lyrics of the song. There are four services available, two for either category, namely $\{S_{11}, S_{12}\}$ or $\{S_{21}, S_{22}\}$. Then, each service has an associated adapter, which can be described in terms of micro-operations:

- $A_{11} = \{MO_1 = 1, MO_2 = 5\}$,
- $A_{12} = \{MO_1 = 3, MO_2 = 2\}$,
- $A_{21} = \{MO_1 = 7\}$,
- $A_{22} = \{MO_1 = 1, MO_2 = 1, MO_3 = 1\}$.

To determine which alternative fits the energy requirements of the developer, we apply the model by adding the energy cost of the micro-operations in order to calculate the estimated energy cost of invoking each service:

— $\{S_{11} = 1 + 5 = 6\}$, $\{S_{12} = 3 + 2 = 5\}$ – $\{S_{21} = 7\}$, $\{S_{22} = 1 + 1 + 1 = 3\}$.

Finally, we calculate the estimated energy cost of each mashup alternative:

— $M_1 = \{S_{11} = 6, S_{21} = 7\} = 13$,
— $M_2 = \{S_{12} = 5, S_{21} = 7\} = 12$,
— $M_3 = \{S_{11} = 6, S_{22} = 3\} = 9$,
— $M_4 = \{S_{12} = 5, S_{22} = 3\} = 8$.

Then, the least energy-consuming combination is $M_4 = \{S_{12}, S_{22}\}$.

In order to evaluate the feasibility of the proposed model, we considered a set $B_k = \{M_1..M_j\}$ composed by the alternatives that have lower energy consumption than M_k. We created two M_k groups, one corresponding to the estimated energy cost CME and another corresponding to the real energy cost CMR which we have measured from experiments. Then, we compared these groups to find errors between the estimated and the real cases. Next section covers these experiments, and uses this approach to evaluate the estimations provided by the model.

3 Experimental evaluation

This section describes the results of the measurements for energy consumption we made for micro-operations along with a case study in which we applied and evaluated the proposed estimation model.

Before introducing the experiments we will describe the measurement environment. We used a Power Monitor, a tool capable of measuring the power of any device that employs a lithium battery at a frequency of 5 KHz (5,000 times per second). This tool comes along with a software application that shows the Voltage and Amperage measurements, and allows the user to export this information in different formats (e.g., CSV). Fig. 4 shows the experimental setup.

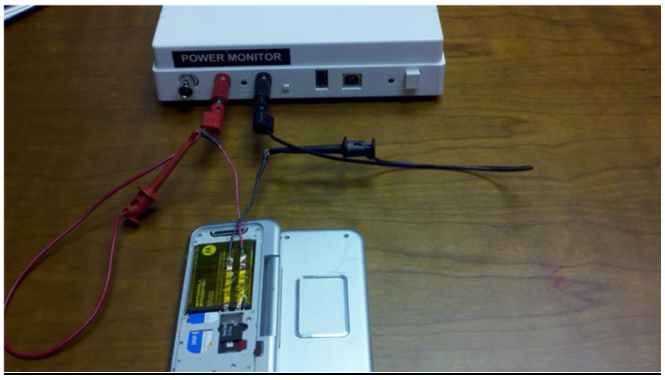

Fig. 4. Power Monitor setup

Additionally, Table 2 shows the specifications of the devices used in the experiments. The experiments were made after a device restart and executing only the essential operating system services.

Table 2. Devices specification

	Samsung I5500	**Samsung I9300**
CPU	600 MHz CPU (model MSM7227-1 ARM11)	Quad-core 1.4 GHz Cortex-A9
RAM	256MB	1GB
Storage	Internal 4GB, uSD 4GB	Internal 16GB, uSD 64GB
Battery	Lithium 1,200 mAh	Lithium 2,100 mAh

3.1 Micro-operations

Earlier, we have presented the concept of micro-operation and how use this to describe Web Service adapters. Also, we have mentioned that a necessary step before applying the model for estimating the overall consumption of a service adapter is to measure the energy consumption of each micro-operation.

Table 3 shows the average energy consumption of several test runs for each micro-operation on both devices. Note that array-related micro-operations had a fixed size. To ensure representativeness, we simulated the creation of arrays of size N. In order to achieve this, we increased the array size in one element every run (N =1 to N=64,000 for array-related micro-operations). The standard deviation for the I9300 on dynamic memory-related micro-operations is somewhat high. This might be associated with the memory usage due to the services that the Samsung software layer adds to the system.

Table 3. Energy consumption results: Micro-operations

	Samsung I5500		**Samsung I9300**	
Micro-operation	**Power consumption (per unit) (µ Joules)**	**Standard Deviation (%)**	**Power consumption (per unit) (µ Joules)**	**Standard Deviation (%)**
MOP1: cast int to double	0.0876	2.2119	0.0257	1.8291
MOP2: cast int to float	0.0833	0.1507	0.0250	1.1715
MOP3: cast long to double	0.6443	0.2524	0.2683	0.1985
MOP4: cast long to float	0.6006	0.2351	0.2708	0.0629

MOP5: create an Object instance	2.2680	1.4066	1.0348	25.4678
MOP6: assign from other instance	0.0216	0.0200	0.0108	0.2862
MOP7: assign from the same instance	0.0199	0.0209	0.0042	0.7279
MOP8: create a float array[n]	49.7047	0.1551	294.5053	15.7948
MOP9: create a int array[n]	49.9425	0.2585	297.5589	8.5689
MOP10: create a long array[n]	97.3300	0.1422	317.5893	32.3114
MOP11: create a Object array[n]	52.1544	0.5370	557.9736	12.8078

Fig. 5 compares micro-operations energy consumption of both I5500 and I9300 devices measured in Joules --the work required to produce one watt of power for one second--. The results indicate that the scale in which different micro-operations differ is similar in the two different devices, e.g. MOP10 consumes almost twice as much energy as MOP8, MOP9 or MOP11 consume in both cases. Also, array creation micro-operations consume more energy than any other micro-operation (about 4,800% in the worst case).

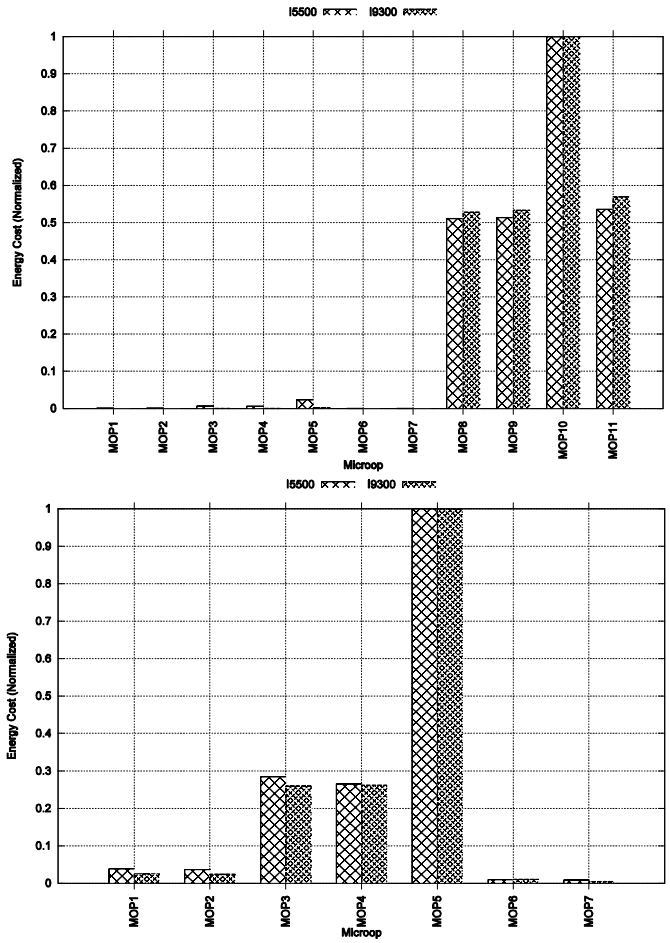

Fig. 5. Energy consumption comparison: Micro-operations. The graphic on bottom is derived from the graphic on the top by excluding MOP8, MOP9, MOP10 and MOP11

Additionally, long array creation is about 100% more energy-hungry than any other array creation micro-operation. This is since the memory size of the long type (64-bits in 32-bit systems) is twice the size of any of the other array creation micro-operations. This means that long arrays micro-operations require twice as much memory for the same array size.

3.2 Case Study

In this section we put in practice the above results to illustrate the utility and feasibility of the proposed estimation model. First, we will present a case study involving the development of a mashup, then we will apply the model to select services, and finally we will compare how accurate the estimations provided by the model are compared with

the real energy consumption of the possible service combinations for building the mashup.

Consider the three services and client-side expected interfaces from Table 4. We created eight mashup versions trough the combination of these services, varying the type and number of parameters. Table 5 shows the micro-operations involved to transform the expected interfaces into the real service interfaces. However, in some cases, the micro-operations proposed cannot describe exactly the operations involved in the adapter, for example, the instantiation of user-defined types. In such cases we used the most similar micro-operations, considering creations of user-defined types as creations of Object types, for instance. Also, by basing on the results presented in Table 3, we applied the model and calculated the estimated cost for each service version, taking into account the micro-operation measurements made on both devices. The size of the arrays in parameters was set to 50 for all services.

Table 4. Client expected interfaces

Service	Expected Interface
Reverse Geocoding	String getAddress(int latitude, int longitude)
Send an E-mail	sendMessage(String text, String subject, String[] recipients)
Send an SMS	sendSMSMessage(String text, String[] recipients)

Table 5. Case study: Adapter estimation results

Service		Adapter micro-operations	Estimated cost (µ Joules)	
Service name	**Version**		**Samsung I5500**	**Samsung I9300**
1: Get a direction	1	• MOP5: 1 • MOP2: 2 • MOP6: 2	2.4778	1.1064
	2	• MOP1: 2 • MOP5: 1 • MOP6: 2	2.4864	1.1078
2: Send an e-mail	1	• MOP5: 1 • MOP6: 52 • MOP9: 1	55.5456	319.1857
	2	• MOP5: 1 • MOP9: 2 • MOP6: 50	103.2330	596.6928
3: Send an SMS	1	• MOP5: 1 • MOP6: 52 • MOP9: 1	55.5456	319.1857
	2	• MOP5: 1 • MOP10: 1 • MOP6: 52	152.8756	877.1593

We measured the real energy consumption of each mashup combination. In order to achieve this, first, we implemented the services presented in Table 5 using Axis2 and the service clients using ksoap2 (SOAP services). Then, we used Apache Tomcat 7 to host the services, running in a server with an Athlon2 X2@2.6Ghz processor, 4 GB RAM, and running Windows 7. The mobile device hosting the client application contacted the server through a Wi-Fi connection to a private access point. In addition, service invocation was made in 10 groups of 1,000 invocations each (10,000 invocations). Also, for each possible service combination, 5 runs were made. As discussed earlier, dynamic memory-related micro-operations presented high deviations in the I9300 device.

Table 6 presents the measurement results. We used the following nomenclature to describe a mashup: (GetDirectionVersion, SendAnEmailVersion, SendAnSMSVersion). According to the results presented in Section 3.1, these results show that array creation micro-operations have a high impact on energy consumption. Also, combinations that involve creating long arrays are significantly more energy-demanding than any other.

Table 6. Energy consumption results: Mashups

	Samsung I5500		**Samsung I9300**	
Mashup	**Energy consumption (per unit) (μ Joules)**	**Standard Deviation (%)**	**Energy consumption (per unit) (μ Joules)**	**Standard Deviation (%)**
M_1**: (1, 1, 1)**	5040.41	4.3760	2590.05	1.3078
M_2**: (1, 1, 2)**	6056.04	3.2016	3128.24	0.3112
M_3**: (1, 2, 1)**	5817.13	3.0445	3244.50	2.7134
M_4**: (1, 2, 2)**	6741.06	2.1450	3618.03	10.5014
M_5**: (2, 1, 1)**	4995.10	0.5860	2735.18	2.0450
M_6**: (2, 1, 2)**	5740.82	3.5858	3330.45	13.2837
M_7**: (2, 2, 1)**	5847.66	1.9406	3285.64	1.5544
M_8**: (2, 2, 2)**	6651.41	2.4424	3593.48	8.2845

Finally, we used the formulas presented in Section 2.3 and the results of the estimations from Table 5 to estimate which mashup combinations have the lowest energy consumption. Fig. 6 shows a comparison between the estimated and the real energy consumption results. The relationship between the real and estimated costs presents high similarity on both devices. This indicates that despite the device, the scale in which different alternatives differ tends to be the same. Additionally, this relationship is a direct consequence of the micro-operation measurements, which also presented the same pattern.

Our model was not intended to provide the exact amount of energy each alternative will actually consume, but to provide energy consumption estimations to EasySOC applications, allowing the developer to select the least expensive mashup combination. Thus, even if there is a large difference between the real and the estimated cost in some cases, this should not affect the results. For example, the mashup (1, 1, 1) combination is less energy-expensive compared to the (1, 1, 2) combination, both in the real and the estimated cases. To evidence this, we used the approach proposed in Section 2.3. First, we created the B_k sets:

- I5500:
  - $B_1 = \{M_5\}$
  - $B_2 = \{M_1, M_3, M_5, M_6, M_7\}$
  - $B_3 = \{M_1, M_5, M_6\}$
  - $B_4 = \{M_1, M_2, M_3, M_5, M_6, M_7, M_8\}$
  - $B_5 = \{\varepsilon\}$
  - $B_6 = \{M_1, M_5\}$
  - $B_7 = \{M_1, M_3, M_5, M_6\}$ • $B_8 = \{M_1, M_2, M_3, M_5, M_6, M_7\}$

- Estimated I5500:
  - $B_1 = \{\varepsilon\}$
  - $B_2 = \{M_1, M_3, M_5, M_7\}$
  - $B_3 = \{M_1, M_5\}$
  - $B_4 = \{M_1, M_2, M_3, M_5, M_6, M_7\}$
  - $B_5 = \{M_1\}$
  - $B_6 = \{M_1, M_2, M_3, M_5, M_7\}$
  - $B_7 = \{M_1, M_3, M_5\}$
  - $B_8 = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7\}$

- I9300:
  - $B_1 = \{\varepsilon\}$
  - $B_2 = \{M_1, M_5\}$
  - $B_3 = \{M_1, M_2, M_5\}$
  - $B_4 = \{M_1, M_2, M_3, M_5, M_6, M_7, M_8\}$
  - $B_5 = \{ M_1\}$
  - $B_6 = \{M_1, M_2, M_3, M_5\}$
  - $B_7 = \{M_1, M_2, M_3, M_5, M_6\}$
  - $B_8 = \{M_1, M_2, M_3, M_5, M_6, M_7\}$

- Estimated I9300:

- $B_1 = \{\varepsilon\}$
- $B_2 = \{M_1, M_3, M_5, M_7\}$
- $B_3 = \{M_1, M_5\}$
- $B_4 = \{M_1, M_2, M_3, M_5, M_6, M_7, M_8\}$
- $B_5 = \{M_1\}$
- $B_6 = \{M_1, M_2, M_3, M_5, M_7\}$
- $B_7 = \{M_1, M_3, M_5\}$
- $B_8 = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7\}$

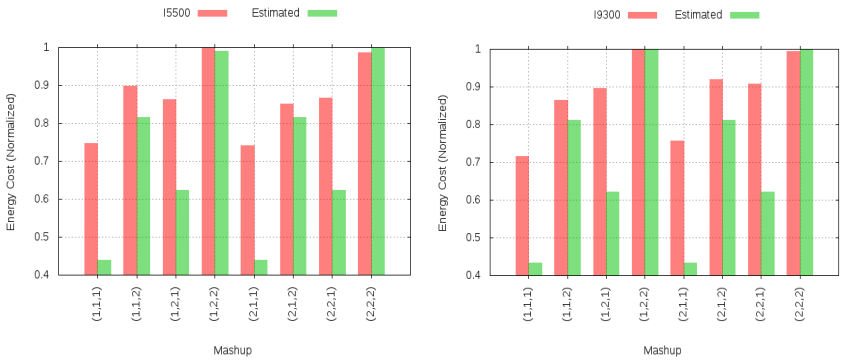

Fig. 6. Energy consumption comparison: Mashups in the I5500 (left) and the I9300 (right)

Then, we compared the differences in both cases. From 27 comparisons made, there were 5 errors for the I5500 and 4 errors for the I9300 (errors are highlighted using red bars). This assessment preliminary shows that the efficacy of the model, for this case study, is $\frac{22}{27} = 0.8148 \approx 81\%$ for the I5500 and $\frac{23}{27} = 0.8518 \approx 85\%$ for the I9300.

4 Related work

Previous works in the area have studied 2G and 3G networks power consumption. The authors in [16] suggest that 2G networks reduce the energy consumption for SMS and voice services, while 3G networks should be the preferred option for network data transfers providing both speed and energy efficiency. Also, [4] proposes TailEnder, a network scheduler that lowers the energy consumption up to 40% in 3G networks by reordering mobile device's network transfers. In the same direction, [11] improves Bit Torrent transfer scheduling, reducing power consumption by almost 50% with respect to conventional clients. Other similar studies have profiled applications network traffic and identified four major application energy leaks: misinterpretation of callback API

semantics, poorly designed downloading schemes, repetitive downloads, and aggressive prefetching [25].

Additionally, other works have studied energy consumption in the context of 302.11 networks, particularly, during the association process [19]. The authors claim that eliminating the ARP probe process from the dynamic addressing protocol could improve dynamic addressing power consumption by almost 400%.

Although the works presented above aim to reduce energy consumption, they are focused on local device's components. A different approach is to delegate or execute parts of the application on external non-mobile devices, reducing application's CPU use and thus, lowering energy consumption [26]. Regardless executing tasks on external devices seems to reduce the amount of energy consumed by an application, this involves data transmission, which in turn creates a tradeoff between the data and the computation being transferred to the external device. In this line, [27] analyzes this relationship by taking into account the task's computation time (TE), the data being transferred (DT), the data being received (DR), and the upload and download bandwidth (AB and BR, respectively). They claim that offloading a particular task would be worth the price when the time required for processing the task in the device is higher than $T = DT / AB + DR / BR + TE$.

On the other hand, the implications of using SOC in mobile applications have been gained attention in the research community. [14] studies service invocation patterns claiming that in 3G networks, an effective service invocation scheduling could improve up to 50% the energy consumption. Other works analyzed proxy-based architectures for service invocation that reduce the intra-device CPU processing during the service invocation process, resulting in a reduction of energy consumption. The authors in [18] suggest two different architectures for invoking services using WSA and SMS. In the same direction [24] proposes a platform that relies on the Jini Surrogate Architecture Specification. In this context, each service has a surrogate which runs in a server connected to the Internet. Then, when a client tries to invoke a service, the invocation is done through the corresponding surrogate, which is the responsible for the interaction with the service. Finally the authors in [3] combine proxy based invocation and offloading improving up to 100% the service's response latency.

Among the works in this line, however, there is no evidence of an approach for service-oriented mobile application development which considers energy consumption as a first-class citizen.

5 Conclusions

In this work we have presented a SOC development model for mobile devices that improves EasySOC by providing energy consumption estimations. This model helps to determine which services or mashup combinations are less energy-hungry, reducing the energy consumption during the service invocation process after the applications have been built and deployed.

First, we have described the EasySOC [7,13] approach for developing SOC applications. Our power-aware enhanced model builds on the benefits of EasySOC and improves this by estimating the energy consumption due to interface adaptations. In order to achieve this, we have proposed to describe service adapters in terms of elemental operations called *micro-operations*. Then, to calculate the overall energy consumption involved in an adaptation, the energy cost of individual micro-operations resulting from describing the adapter are summed. This is in turn used to estimate which mashup combinations require less energy.

To assess our approach we measured the energy consumption of each individual micro-operation on two different devices, Samsung I5500 and Samsung I9300. We identified that array micro-operations are more energy-hungry than any others and that creation of larger data-types increment energy consumption. This suggests it could be a relationship between energy consumption and larger memory requirements. Second, using as an input the energy cost of the micro-operations provided by the measurements, we described a case study in which we applied the proposed model. Also, we have validated the model comparing the estimations against the real measurements. The results indicated that the model had an efficacy around 81-85% for the proposed case study.

This work can be extended in several directions. Firstly, we will generalize the proposed model by incorporating to the catalog micro-operations that help to describe more precisely service adapters, for example by considering array assignation and common programming language defined-types such as strings or char. Also we will study the tradeoffs between (partially) including vs. not including service adapters in applications, and how this impacts on energy consumption, maintainability, and performance. Indeed, a previous study in the context of EasySOC [13] shows the trade-off between performance and maintainability when not using adapters, but energy should be considered alongside these two attributes.

On the other hand, we will experiment with new device types (e.g., tablets) and new smartphones to further generalize the measurements results. Additionally, we will extend WSQBE [8] by providing real energy estimations for invoking services. WSQBE is an approach that combines popular best practices for using external Web services with text-mining and machine learning techniques for facilitating service publication and discovery. Thus, when a client searches for a service by providing the expected interface, WSQBE will return a similar service in functional terms but at the same time the least energy hungry one according to the amount of energy that adaptations will consume. Finally, we will extend our model to support REST Web Services [5,9], and we will analyze the trade-offs against SOAP services.

Acknowledgments

We acknowledge the financial support of ANPCyT through grants PAE-PICT-2007-2311 and PICT-2012-0045.

References

1. Yuvraj Agarwal, Curt Schurgers, and Rajesh Gupta. Dynamic power management using on demand paging for networked embedded systems. ASP-DAC ’05 Proceedings of the 2005 Asia and South Pacific Design Automation, 2005.
2. Mauricio Arroqui, Cristian Mateos, Claudio Machado, and Alejandro Zunino. Restful web services improve the e_ciency of data transfer of a whole-farm simulator accessed by android smartphones. Computers and Electronics in agriculture, 87:14–18, 2011.
3. Hassan Artail, Kassem Fawaz, and Ali Ghandour. A proxy-based architecture for dynamic discovery and invocation of web services from mobile devices. Services Computing, IEEE Transactions, 5:99 – 115, 2012.
4. Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In IMC ’09 Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, pages 280–293, 2009.
5. Robert Battle and Edward Benson. Bridging the semantic web and web 2.0 with representational state transfer. Web Semantics: Science, Services and Agents on the World Wide Web, 2008.
6. Martin Bichler and Kwei-Jay Lin. Service-Oriented Computing. Computer, 39(3):99–101, 2006.
7. Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo. Easysoc: Making web service outsourcing easier. Information Sciences, (0), 2014.
8. Marco Crasso, Alejandro Zunino, and Marcelo Campo. Combining query-byexample and query expansion for simplifying Web Service discovery. Information Systems Frontiers, 13:407–428, 2011.
9. Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, 2000.
10. B. Flipsen, J. Geraedts, A. Reinders, C. Bakker, I. Dafnomilis, and A. Gudadhe. Environmental sizing of smartphone batteries. In Electronics Goes Green 2012+(EGG), 2012, pages 1–9, 2012.
11. Imre Kelényi and Jukka K. Nurminen. Bursty content sharing mechanism for energylimited mobile devices. In PM2HW2N ’09 Proceedings of the 4th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks, 2009.
12. Li Luqun, Li Minglu, and Cui Xianguo. The study on mobile phone-oriented application integration technology of web services. Grid and Cooperative Computing, 3032:867–874, 2004.
13. Cristian Mateos, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Separa-tion of concerns in service-oriented applications based on pervasive design patterns. In SAC ’10 Proceedings of the 2010 ACM Symposium on Applied Computing, 2010.
14. Apostolos Papageorgiou, Ulrich Lampe, Dieter Schuller, Ralf Steinmetz, and Athanasios Bamis. Invoking web services based on energy consumption models. In Mobile Services (MS), 2012 IEEE First International Conference on, 2012.
15. Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. Pervasive Computing, IEEE, 2005.
16. Gian Paolo Perrucci, Frank H.P. Fitzek, Giovanni Sasso, Wolfgang Kellerer, and Jšorg Widmer. On the impact of 2g and 3g network usage for mobile phones battery life. In Wireless Conference, 2009. EW 2009. European, pages 255 – 259, 2009.

17. Vijay Raghunathan, Trevor Pering, Roy Want, Alex Nguyen, and Peter Jensen. Experience with a low power wireless mobile computing platform. ISLPED '04 Proceedings of the 2004 international symposium on Low power electronics and design, 2004.
18. Rendon, Pabon, Vargas, and Guaca. Architectures for web services access from mobile devices. In Web Congress, 2005. LA-WEB 2005. Third Latin American, 2005.
19. Andrew Rice and Simon Hay. Measuring mobile phone energy consumption for 802.11 wireless networking. Pervasive and Mobile Computing, Volume 6, Issue 6, 6:593–606, 2010.
20. Ana Rodriguez, Cristian Mateos, and Alejandro Zunino. Mobile device-aware refactorings for computational kernels. In 13th Argentine Symposium on Technology, 41st JAIIO, 2012.
21. Juan M. Rodriguez, Cristian Mateos, and Alejandro Zunino. Are smartphones really useful for scientific computing? In F. V. Cipolla-Ficarra et al., editor, Advances in New Technologies, Interactive Interfaces and Communicability (ADNTIIC 2011), Lecture Notes in Computer Science, n/a:35–44, 2011.
22. Juan Manuel Rodriguez, Alejandro Zunino, and Marcelo Campo. Introducing mobile devices into grid systems: A survey. International Journal of Web and Grid Services, 7(1):1–40, 2011.
23. Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In Proceedings of the 21st international conference on World Wide Web, 2012.
24. Aart van Halteren and Pravin Pawar. Mobile service platform: A middleware for nomadic mobile service provisioning. In Wireless and Mobile Computing, Networking and Communications, 2006. (WiMob'2006). IEEE International Conference on, 2006.
25. Lide Zhang, Mark S. Gordon, Robert P. Dick, Z. Morley Mao, Peter Dinda, and Lei Yang. Adel: an automatic detector of energy leaks for smartphone applications. In CODES+ISSS '12 Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2012.
26. Kumar, Karthik, and Yung-Hsiang Lu. "Cloud computing for mobile users: Can offloading computation save energy?." *Computer* 4 (2010): 51-56.
27. Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. IEEE, 14:1232 – 1243, 2011.