# Un modelo de desarrollo de aplicaciones orientadas a servicios eficientes en batería para dispositivos móviles

17 de septiembre de 2013


- Alumno: Ignacio Lizarralde
- Director: Dr. Alejandro Zunino
- Co-Director: Dr. Cristian Mateos Díaz


## 1. Introducción

Las personas se encuentran cada vez más enfrentadas a una creciente incorporación en la vida diaria de la computación móvil. Ésta representa una rama de la computación distribuida que estudia aquellos sistemas de información en los que la ubicación de los componentes de hardware y software puede cambiar en forma dinámica. Respecto del primer tipo de componentes, con la popularización de las redes inalámbricas y el surgimiento de dispositivos de naturaleza personal y móvil tales como smartphones y tablets, se ha generado una línea de hardware contemporáneo que ofrece conocidas ventajas respecto a las PC. Estos dispositivos permiten a los usuarios realizar tareas y acceder a información y servicios remotos a cualquier hora y desde cualquier lugar. Esto ha generado una incesante demanda de dispositivos móviles que soportan un amplio rango de aplicaciones de usuario. Acompañando esta tendencia, se han manifestado dos cambios importantes, uno relacionado al hardware en sí de los dispositivos, y el otro asociado a la forma en que se construye el software que corre en éstos.

Por un lado, los dispositivos en los últimos años han evolucionado pasando de ser limitadas agendas electrónicas y medios de navegación a poderosas y pequeñas computadoras [21]. Si bien esto implica que los dispositivos móviles poseen hoy capacidades de cómputo que incluso pueden ser útiles en ambientes de procesamiento distribuido tales como Grids [18], lo cual era antes impensado, se pone de manifiesto el compromiso que surge cuando se aprovechan dichas capacidades: el tiempo de vida de batería se ve afectado negativamente de forma significativa. Las capacidades de las baterías de dichos dispositivos no han acompañado el crecimiento de la energía demandada del hardware [10]. Así, por ejemplo, mientras hace 5 años la batería de un smartphone duraba hasta una semana, en los dispositivos actuales incluso no alcanza para satisfacer las demandas de una jornada laboral [10]. Si bien comparar dispositivos de hace algunos años con dispositivos actuales no es justo por diversos motivos (pantallas de mayor tamaño, software más complejo, mayor tiempo de utilización, etc.), es evidente que una de las problemática más importantes que enfrenta el área está relacionada con la duración de batería.

Por otro lado, mientras que el software móvil de hace algunos años mayormente ejecutaba en los dispositivos de forma mayormente aislada (esto es, intra-dispositivo y sin necesidad de red), actualmente el software hace uso intensivo de las radios de los dispositivos móviles, a través de conexiones

3G o WiFi, para acceder a recursos de Internet mediante Servicios Web [1, 23]. Esto tiene un impacto muy importante en la duración de batería, pues junto con el CPU, las radios son los elementos que mayor cantidad de energía consumen. En definitiva, el software móvil actual depende para su ejecución de una conexión de red, pues las aplicaciones móviles son distribuidas por naturaleza, en el sentido de que normalmente acceden a recursos de servidores mediante Internet [14]. Este tipo de software, también denominado "mashup" de servicios, consiste en aplicaciones móviles nativas que consumen contenido remoto a través de Servicios Web. Esta clase de aplicaciones se origina en el área de computación distribuida, y particularmente en las sub-áreas de Grid y Cloud Computing, que estudian cómo construir ambientes distribuidos que agrupan recursos de hardware (CPUs, memoria, discos) de los servidores que los componen para dotar a las aplicaciones de usuario de vastas capacidades computacionales [11].

En esencia, el modelo canónico de Servicios Web [15] prescribe la existencia de piezas de software reutilizables que se ofrecen a través de la Web, las cuales constan de una especificación funcional legible por el humano, y una implementación "server-side" que ejecuta en un servidor. El modelo además prescribe la existencia de registros que ofician de páginas amarillas y almacenan la información de los servicios disponibles. Así, por ejemplo, desarrollar una aplicación móvil típica, esto es, que realiza un "mashup" con varios servicios remotos requiere a) descubrir los servicios necesarios, b) seleccionar de entre varios candidatos que provean la misma funcionalidad o pieza de información, y c) finalmente integrar los mismos en la aplicación mediante la utilización de patrones de diseño clásicos, normalmente Proxy y Adapter.

En desarrollo orientado a servicios (SOC) tradicional, esto es, en entornos de escritorio, se ha encontrado que el modelo de desarrollo *code-first* de lado cliente [7], según el cual el desarrollador primero genera una pre-especificación de los servicios necesarios antes de que los pasos a)-c) tengan lugar, acelera el descubrimiento y selección de servicios, y al mismo tiempo resulta en aplicaciones cliente menos acopladas con los servicios seleccionados. Estas ventajas ingenieriles, sin embargo, pueden ser susceptibles de inconvenientes cuando se las considera en el contexto de aplicaciones móviles. Este modelo de desarrollo hace énfasis en encontrar eficientemente los servicios disponibles más acordes a las necesidades funcionales de los usuarios, pero sin tener en cuenta el costo en términos de adaptación de las pre-especificaciones de los servicios con la interfaz real expuesta por éstos últimos, la cual se realiza en el dispositivo. Como es de vital importancia alargar lo más posible la capacidad de batería por cuestiones de prestación, es necesario un modelo de desarrollo SOC que tenga en cuenta tanto aspectos funcionales tradicionales como consumo de batería. Ciertamente, los esfuerzos por medir objetivamente y optimizar la eficiencia de dicho modelo de desarrollo SOC se han centrado en mejorar la eficiencia de los registros de servicios [12, 24, 6] (utilizando métricas del área de Recuperación de Información tales como Precision-at-n y Recall) y estimar la integrabilidad funcional de servicios requeridos y disponibles [5] para acelerar el desarrollo. Existe, sin embargo, la necesidad de incorporar como métrica de eficiencia la cuantificación del consumo de batería de las aplicaciones SOC resultantes.

## 2. Objetivos

El objetivo específico es entonces extender el modelo de desarrollo SOC detallado en [7] para su empleo en el contexto de aplicaciones móviles. En pos de esto, desde el punto de vista de un dispositivo móvil accediendo a Servicios Web, se desarrollarán mecanismos de búsqueda y selección de servicios que permitan localizar dentro de un registro los servicios que se estima resultarán más baratos de acceder en términos del uso de CPU y ancho de banda en el dispositivo, pero que cumplen

con requerimientos funcionales dados por el desarrollador. En otras palabras, dada una serie de Servicios Web candidatos para una funcionalidad específica (por ejemplo localizar un componente que provea operaciones estadísticas complejas o acceso a una fuente de datos) los mecanismos de búsqueda y selección tendrán en cuenta el costo estimado de uso de los servicios en término de batería y red, al mismo tiempo garantizando niveles aceptables de integrabilidad [5].

Para esto, se extenderán los registros de Servicios Web convencionales para ofrecer un mejor balance entre la relevancia funcional y costo de invocación de los Servicios Web retornados como candidatos ante una consulta. Se tomará como punto de partida WSQBE+ [8], un registro para el almacenamiento y búsqueda de Servicios Web desarrollado recientemente. WSQBE+ se basa en el uso de técnicas provenientes del área de Machine Learning tales como *text mining*, *word stemming* y *word disambiguation*, y provee una efectividad muy superior a otros buscadores similares en términos de métricas del área de Recuperación de Información [8].

Entre los aspectos a considerar al extender WSQBE+, esencialmente se encuentra el de dotar al registro con estimaciones fieles de la utilización de CPU y batería de los clientes de servicios en dispositivos móviles para invocar servicios. Se considerarán como punto de partida [17] y [16], trabajos que exploran el consumo de batería de códigos Java intensivos en CPU a través de benchmarks estándar y microbenchmarks. Estos trabajos ahondan en la problemática de identificar las prácticas de programación más costosas en términos de batería, avance que será adaptado para determinar las operaciones de representación e intercambio de datos más costosas al interactuar con servicios desde un dispositivo en aplicaciones de tipo “mashups”.

A continuación se discuten antecedentes relacionados con el trabajo propuesto.

## 3. Antecedentes

Una problemática actual en desarrollo móvil la constituye la detección de patrones de programación y diseño que afecten negativamente el uso de batería, y eventualmente formas de detectar automáticamente estos patrones de forma supervisada y semi-supervisada. Con respecto a la optimización de código, existen diversos trabajos que estudian la performance de distintos patrones. Por ejemplo, en [4] se compara código Java y código nativo en un dispositivo Android. Es importante hacer la distinción de que el código Java no corre en una máquina virtual Java, sino que Android implementa otra máquina virtual conocida como Dalvik. Esta máquina interpreta y compila en tiempo de ejecución el byte-code Dalvik, por lo que en líneas generales la el funcionamiento es similar a una máquina virtual Java tradicional. Se realizaron 12 tests para evaluar la performance y se encontró que el código nativo es más rápido que el código Java en un 34,2 %. En [22] se muestra la diferencia en términos de performance entre una aplicación para Android utilizando código nativo y una aplicación usando Java. Como conclusión, se recomienda utilizar bibliotecas nativas C cuando la aplicación requiere de accesos frecuentes a la memoria y realiza cálculos complejos. Se aprecia entonces un incipiente surgimiento de investigaciones en cuanto a guías de desarrollo para aprovechar adecuadamente dispositivos móviles. Sin embargo, estos análisis sólo se enfocan en velocidad de ejecución y aunque es razonable pensar que menor tiempo de uso de CPU equivale a menor consumo de batería, esto no ha sido probado aún. Además, estos resultados aplican a aplicaciones móviles tradicionales, que no realizan una fuerte interacción con servicios externos.

Existen además dos trabajos [17, 16] preliminares que apuntan a que ciertos patrones de código son energéticamente más eficientes que otros. Ambos trabajos están centrandos en código Java (no “mashup”) corriendo en la plataforma Android. El primer trabajo [17] se limitó a comparar cuánto cómputo intensivo se puede ejecutar en dispositivos Android utilizando una carga de batería contra

la cantidad de computación ejecutadas en diversas notebooks tradicionales bajo las mismas condiciones. Para esto se corrieron distintos benchmarks tradicionales implementados en Java, tanto en las notebooks como en los dispositivos Android. A pesar de no haber sido optimizados, los resultados mostraron que los dispositivos pueden ejecutar comparativamente una cantidad de computación significativa antes del agote de batería. En [16] se estudiaron distintas optimizaciones de código para mejorar la eficiencia energética. La manera de evaluar consistió en ejecutar el código sin optimizar y el código optimizado hasta agotar la batería. En algunos casos, los códigos optimizados ejecutaron hasta 8 veces más que el código sin optimizar. Las optimizaciones son en general simples, como acceder una variable de instancia mediante un método "get" o directamente haciéndola pública. En las adaptaciones al invocar servicios, sin embargo, existen una cantidad importante de patrones por analizar que escapan a estas conclusiones, relacionados con formateo de datos y transferencia de los mismos por red, usados en los adaptadores de los "mashups".

El "mashup" de servicios no es un patrón de construcción de aplicaciones exclusivo del desarrollo móvil, sino que tiene sus raíces en el desarrollo de aplicaciones de escritorio con el surgimiento de los Servicios Web. Los Servicios Web son un conjunto de protocolos, originados en la Web, agnósticos de la tecnología de base que permiten a dos aplicaciones comunicarse entre ellas independientemente de en que plataforma ejecute cada aplicación. En este contexto, existen varios trabajos [19, 20, 13, 12, 8, 7, 6] centrados principalmente en asistir a los desarrolladores de aplicaciones de escritorio que utilizan o proveen Servicios Web, para lo cual se hace énfasis en tres actividades claves: publicación de los Servicios Web en registros públicos, búsqueda de Servicios Web en los mismos, y consumo de los servicios. Puntualmente, en [8] se presentó un buscador de Servicios Web cuyo objetivo es facilitar tanto la publicación como el descubrimiento de los mismos. Para la publicación, sólo se requiere alimentar al buscador con la definición de la interfaz del Servicio Web escrita en el lenguaje WSDL. En cuanto a la búsqueda, el buscador utiliza la mínima pieza de documentación acerca de un componente –que es su interfaz de programación– como pre-especificación o consulta con el objetivo de buscar los Servicios Web con una interfaz similar a la esperada.

Finalmente, una vez que el desarrollador encuentra un Servicio Web que satisfaga sus necesidades debe adaptar su aplicación para consumirlo. La adaptación puede ser un proceso laborioso y debe repetirse cada vez que se desee reemplazar el Servicio Web por uno similar. Para solucionar este problema, se desarrolló un plug-in[1] para el entorno de desarrollo Java Eclipse. La herramienta soporta la transformación de aplicaciones convencionales para aprovechar Servicios Web [7] permitiendo componer distintos servicios sin mucho esfuerzo.

Con respecto a la utilización de Servicios Web en dispositivos móviles, a destacar está un estudio [1] sobre la eficiencia energética de distintos frameworks para consumir Servicios Web, acotado al dominio de aplicaciones móviles agropecuarias. Además de que el trabajo muestra que los Servicios Web en dispositivos móviles son altamente relevantes para aplicaciones reales del sector productivo, del trabajo se obtuvo experiencia en cuanto a guías tecnológicas para el consumo eficiente en batería de Servicios Web teniendo el cuenta el factor de tipo de red inalámbrica interviniente. Ciertamente, existe un creciente interés en el área de Servicios Web en cuanto al desarrollo de soportes energéticamente eficientes. Un ejemplo emblemático es [2], donde se presenta un modelo para estimar consumo de energía en base a carga de trabajo de un Servicio Web y características del hardware sobre el cual se ejecuta. Los autores concluyen que conociendo el hardware se podría ejecutar el servicio y estimar un modelo de consumo de energía con bajo porcentaje de error. Sin embargo, este tipo de trabajos apuntan a mejorar la eficiencia energética dentro de un datacenter independientemente del dispositivo invocador de los servicios.

[1] https://sites.google.com/site/easysoc/home/service-adapter

Así como estimar el potencial consumo de energía de un Servicio Web permite hacer un uso más eficiente de la energía en los servidores en los proveedores de servicios, es de relevancia investigar la hipótesis de que la elección e incorporación de un Servicio Web con una interfaz particular también incide en la utilización de la batería de los dispositivos de los clientes, particularmente los dispositivos móviles. Esta hipótesis se sustenta en que por ejemplo las aplicaciones clientes que se ejecutan sobre el sistema operativo Android, o incluso las de Windows Mobile, modelan sus componentes internos de una manera que para poder interactuar con la interfaz de un servicio externo se deben realizar operaciones de adaptación y formateo de datos. A modo de ejemplo, una operación de adaptación consiste en cambiar la representación de un dato desde un arreglo de caracteres en UTF-8 a uno en UTF-16. Claramente, en un contexto heterogéneo es muy común que este u otros tipos de adaptaciones deban realizarse para consumir efectivamente un servicio particular. Dado que a mayor cantidad de operaciones de adaptación, mayor puede ser el consumo de energía. Por lo tanto, investigar cómo impactan en la vida de las baterías de los dispositivos las operaciones de adaptación y patrones de uso de red, para dotar a los desarrolladores de aplicaciones móviles de criterios de selección o mecanismos de descubrimiento e integración de servicios basados en dicho impacto, es un problema muy actual.

# 4. Actividades y metodología

Como pilar fundamental del trabajo a desarrollar, vale recalcar los estudios exhaustivos ya existentes en torno a modelos de programación de aplicaciones SOC desde el punto de vista del cliente [18], de los registros de servicios [9], y del proveedor de servicios [13]. A destacar en la misma línea se encuentran desarrollos recientes relacionados con el estudio de la eficiencia energética en tablets y smartphones [16, 17, 18], los cuales serán tomados como base para la confección de modelos para estimar el costo energético de los patrones de interacción y uso de red en la interacción dispositivo-servicio.

En este sentido, la metodología para alcanzar el objetivo específico planteado descansará en una combinación de elaboración de modelos de estimación de costo de invocación de servicios (y su incorporación a WSQBE+) por un lado, y experimentación real por otro. Es necesario a grandes rasgos estimar qué tan costoso resulta en términos de batería la utilización de diferentes proxies a servicios de lado cliente, y asimismo el impacto en la integrabilidad de los servicios asociados. Para estimar costo de batería, es necesario modelar el universo de proxies posible y cuantificar el costo tecnológico de uso tanto intra-dispositivo como en términos de uso de red.

Para los experimentos en sí, se cuenta con varios dispositivos móviles, financiados por los proyectos PAE-PICT-02311 y PICT-2012-0045 de ANCPyT. Cabe destacar que para acotar el alcance del trabajo, se utilizarán dispositivos móviles basados en el sistema operativo Android. Esta elección se fundamenta en la importante cantidad de dispositivos Android existentes en el mundo. De acuerdo a los anuncios de Google I/O, ya en el 2011 existían 310 modelos de smartphones basados en Android producidos por 36 fabricantes [17]. Además, Android es un sistema operativo abierto que permite adaptar y extender muchas de sus características internas [3], lo que facilita la experimentación. Además, también en el contexto del proyecto PAE-PICT-02311, se cuenta con un equipo de medición de baterías de litio a nivel de hardware (`http://www.msoon.com/LabEquipment/PowerMonitor/`), que permite obtener mediciones con un nivel de precisión muy superior a los esquemas basados en software. Debido a las características de este tipo de baterías, al saber el voltaje y los miliamperios de una batería se puede saber cuál es su consumo en un intervalo de tiempo [25].

De esta forma, dados dos servicios candidatos para una misma funcionalidad requerida en un "mashup", será posible medir de forma precisa el costo energético producto del intercambio y forma-

teo de datos con uno u otro candidato. Así, será posible construir modelos empíricos que permitan estimar con exactitud aceptable el costo de invocación de servicios, incorporando esta información al criterio de ranking actual de los buscadores de servicios. Los mecanismos y modelos desarrollados se validarán mediante escenarios realistas de consumo y provisión de Servicios Web, para evaluar los respectivos ahorros en términos de batería.

# 5. Cronograma

A continuación se detalla el cronograma de actividades y tiempos estimados. El tiempo total se ha estipulado en 7 (siete) meses, pero vale la pena aclarar que para varias actividades ya se han producido avances importantes.

| **Cronograma de actividades** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actividad** | **Meses** | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Relevamiento y análisis bibliográfico | X | X | | | | | | | | | | |
| Diseño de benchmarks de batería para dispositivos Android en escenarios de uso de CPU y red representativos (mashups) | | X | | | | | | | | | | |
| Ejecución de Benchmarks en Smartphones | | X | X | | | | | | | | | |
| Análisis de los resultados de los Benchmarks para determinar patrones de consumo batería | | | X | X | | | | | | | | |
| Modelado de proxies | | | | X | X | | | | | | | |
| Incorporación de información de consumo de batería al modelo code-first lado cliente SOC actual | | | | | X | | | | | | | |
| Experimentación y análisis de resultados | | | | | X | X | | | | | | |
| Redacción del informe final | | | | | X | X | X | | | | | |

# Referencias


[1] Mauricio Arroqui, Cristian Mateos, Claudio Machado, and Alejandro Zunino. RESTful web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones. *Computers and Electronics in Agriculture*, 87(0):14–18, 2012.

[2] Peter Bartalos, M. Brian Blake, and Sekou Remy. Green web services: Models for energy-aware web services and applications. *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, 0:1–8, 2011.

[3] Sören Blom, Matthias Book, Volker Gruhn, Ruslan Hrushchak, and André Köhler. Write once, run anywhere - a survey of mobile runtime environments. In *3rd International Conference on Grid and Pervasive Computing*, GPC-WORKSHOPS '08, pages 132–137, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Lin Cheng-Min, Lin Jyh-Horng, Dow Chyi-Ren, and Wen Chang-Ming. Benchmark dalvik and native code for android system. In *Second International Conference on Innovations in Bio-inspired Computing and Applications*, 2011.

[5] M. Crasso, C. Mateos, A. Zunino, and M. Campo. Aasii: A novel algorithm for assessing services interfaces integrability. In *XIII Argentine Symposium on Software Engineering (ASSE2012) - 41 JAIIO*. SADIO, 2012.

[6] M. Crasso, A. Zunino, and M. Campo. Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, 71(2):144–164, 2008.

[7] Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo. Easysoc: Making web service outsourcing easier. *Information Sciences*, (0), 2010. En prensa.

[8] Marco Crasso, Alejandro Zunino, and Marcelo Campo. Combining query-by-example and query expansion for simplifying web service discovery. *Information Systems Frontiers*, 13(3):407–428, 2011.

[9] Marco Crasso, Alejandro Zunino, and Marcelo Campo. A survey of approaches to Web Service discovery in Service-Oriented Architectures. *Journal of Database Management*, 22(1):103–134, 2011.

[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] Ian Foster. The Grid: Computing without bounds. *Scientific American*, 288(4):78–85, 2003.

[12] C. Mateos, M. Crasso, A. Zunino, and J. Ordiales Coscia. Revising wsdl documents: Why and how - part ii. *IEEE Internet Computing*, 2013. En prensa.

[13] Cristian Mateos, Marco Crasso, Alejandro Zunino, and José Luis Ordiales Coscia. Detecting WSDL bad practices in code-first Web Services. *International Journal of Web and Grid Services*, 7:357–387, 2011.

[14] Tommi Mikkonen and Antero Taivalsaari. Cloud computing and its impact on mobile software development: Two roads diverged. *Journal of Systems and Software*, 2013. In press.

[15] Michael Papazoglou and Willem-Jan van den Heuvel. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2(4):412–442, 2006.

[16] Ana Rodriguez, Cristian Mateos, and Alejandro Zunino. Mobile devices-aware refactorings for scientific computational kernels. In *13th Argentine Symposium on Technology. 41st JAIIO*, pages 61–72, 2012.

[17] 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, pages 35–44. Springer, 2011.

[18] Juan Manuel Rodriguez, Marco Crasso, Cristian Mateos, and Alejandro Zunino. Best practices for describing, consuming, and discovering web services: a comprehensive toolset. *Software: Practice and Experience*, pages n/a–n/a, 2012. En prensa.

[19] Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Automatically detecting opportunities for Web Service descriptions improvement. In Wojciech Cellary and Elsa Estevez, editors, *Software Services for e-World*, volume 341 of *IFIP Advances in Information and Communication Technology*, pages 139–150. SADIO - IFIP, Springer Boston, 2010.

[20] Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Improving web service descriptions for effective service discovery. *Science of Computer Programming*, 75(11):1001–1021, 2010.

[21] 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.

[22] Lee Sangchul and Jeon Jae Wook. Evaluating performance of android platform using native c for embedded systems. In *International Conference on Control, Automation and Systems*, 2010.

[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*, WWW '12, pages 41–50, New York, NY, USA, 2012. ACM.

[24] Chen Wu. WSDL term tokenization methods for ir-style web services discovery. *Science of Computer Programming*, 77(3):355–374, 2012.

[25] Lide Zhang, B. Tiwana, R.P. Dick, Zhiyun Qian, Z.M. Mao, Zhaoguang Wang, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 105–114. IEEE/ACM/IFIP, October 2010.