Optimizing Mobile Application Performance with Model-Driven Engineering

Chris Thompson, Jules White, Brian Dougherty, and Douglas C. Schmidt

Department of Electrical Engineering and Computer Science,

Vanderbilt University, Nashville, TN USA

{jules, briand, schmidt}@dre.vanderbilt.edu &
chris.m.thompson@vanderbilt.edu

Abstract. Future embedded and ubiquitous computing systems will operate continuously on mobile devices, such as smartphones, with limited processing capabilities, memory, and power. A critical aspect of developing future applications for mobile devices will be ensuring that the application provides sufficient performance while maximizing battery life. Determining how a software architecture will affect power consumption is hard because the impact of software design on power consumption is not well understood. Typically, the power consumption of a mobile software architecture can only be determined after the architecture is implemented, which is late in the development cycle when design changes are costly.

Model-driven Engineering (MDE) is a promising solution to this problem. In an MDE process, a model of the software architecture can be built and analyzed early in the design cycle to identify key characteristics, such as power consumption. This paper describes current research in developing an MDE tool for modeling mobile software architectures and using them to generate synthetic emulation code to estimate power consumption properties. The paper provides the following contributions to the study of mobile software development: (1) it shows how models of a mobile software architecture can be built, (2) it describes how instrumented emulation code can be generated to run on the target mobile device, and (3) it discusses how this emulation code can be used to glean important estimates of software power consumption and performance.

1 Introduction

Emerging trends and challenges. Mobile devices, such as smartphones, mobile internet devices and web-enabled media players, are becoming pervasive. These devices possess limited resources, such as battery capacity, which requires developers to carefully manage resource consumption. To optimize resource utilization, mobile application developers must understand the trade-offs between performance and bat-

tery life. It is hard to predict the effects of architectural optimizations in mobile devices until a system has been completely implemented, which makes it difficult to test power consumption and performance until late in the software lifecycle [14], e.g., during implementation and testing. Changes made at this point usually result in far-reaching consequences to the overall design and cost much more compared to those made during earlier software lifecycle phases [12], e.g., during architectural design and analysis.

Conventional techniques for developing mobile device software are not well-suited to identifying performance and power consumption trade-offs during earlier phases of the software lifecycle. These limitations stem largely from the difficulty of comparing the power consumption of one architectural design versus another without implementing and testing each on the target device. Moreover, for each function an application performs, there are often multiple possible designs for accomplishing the same task, each differing in terms of operational speed, battery consumption and accuracy. Even though these design variations can significantly impact device performance, there are too many permutations to implement and test each.

For example, if a mobile application communicates with a server it can do so via several protocols, such as HTTP, HTTPS, or other socket connections. Developers can also elect to have the application and/or mobile device infrastructure submit data immediately or in a batch at periodic intervals. Each design option can result in differing power consumption profiles [13]. If the developer elects to use HTTPS over HTTP, the developer is provided with additional security. The overhead associated with key exchange and the encryption/decryption process, however, incurs additional processing time and increases the amount of information that must be transmitted over the network. Both of these require more power and time than would be required if standard HTTP was used.

The combination of these architectural options results in too many possible variations to implement and test each one within a reasonable budget and production cycle. A given application could have hundreds or thousands of viable configurations that satisfy the stated requirements.

Solution approach → *Emulation of application behavior through model-driven testing and auto-generated code*. Model-driven engineering (MDE) [15] provides a promising solution to the challenges described above. MDE relies on modeling languages, such as domain-specific modeling languages (DSMLs) [16], to visually represent various aspects of application and system design. These models can then be utilized for code generation and performance analysis. By creating a model of candidate solution architectures early in the design phase, instrumented architectural emulation code can be generated and then run on actual mobile devices. This MDE-based approach allows developers to quickly emulate a multitude of possible configurations and provides them with actual device performance data without investing the time and effort manually writing application code.

The generated code emulates the modeled architecture by consuming sensor data, computational cycles, and memory as specified in the model, as well as transmitting/receiving faux data over the network. Since wireless transmissions consume most

of the power on mobile devices [3] and network interaction is a key performance bottleneck, large-scale power consumption and performance trends can be gleaned by executing the emulation code. Moreover, as the real implementation is built, the actual application logic can be used to replace faux resource consuming code blocks to refine the accuracy of the model. This MDE-based solution has been utilized previously to eliminate some inherent flaws with serialized phasing in layered systems, specifically as they apply to system QoS and to identify design flaws early in the software production life-cycle [9]. Some prior work [8] also employs model-driven analysis to conduct what-if analysis on potential application architectures.

By utilizing MDE-based analysis, mobile software developers can quantitatively evaluate key performance and power consumption characteristics earlier in the software lifecycle (e.g., at design time) rather than later (e.g., during and after implementation), thereby significantly reducing software refactoring costs due to design flaws. MDE provides this by not only allowing developers to generate emulation code, but also by providing them with a high-level understanding of their application that is easy to modify on the fly. Changes can be made at design time by simply moving model elements around rather than rewriting code. Moreover, since emulation code is automatically generated from the model, developers can quickly understand key performance and power consumption characteristics of potential solution architectures without investing the time and effort to implement them.

This paper describes emerging R&D efforts that seek to provide developers of mobile applications with an MDE-based approach to optimizing application resource consumption across a multitude of platforms at design time. This paper also describes a methodology for increasing battery longevity in mobile devices through application-layer modifications. By focusing on the application layer, developers can still reap the benefits of advanced SDKs and compilers that shield the developer from hardware-centric decisions.

Paper organization. The remainder of this paper is organized as follows: Section 2 presents a sample mobile application running on Google's Android platform and introduces several challenges associated with resource consumption optimization and mobile application development; Section 3 discusses our current research work developing an MDE tool to allow developers to predict software architecture performance and power consumption properties earlier in the development process; Finally, Section 4 presents concluding remarks and lessons learned.

2 Motivating Example

This section presents a motivating mobile application running on Google's Android platform and describes several challenges associated with resource consumption optimization and mobile application development.

2.1 Overview of Wreck Watch

Managing system resources properly can significantly affect device performance and battery life. For instance, reducing CPU instructions not only speeds performance but also reduces the time the process is in a non-idle state thereby reducing power consumption; reducing network traffic also speeds performance and reduces the power supplied to the radio. To demonstrate the importance of proper resource management and the value of model-based resource analysis, we present the following example mobile application, called *Wreck Watch*, shown in Figure 1.

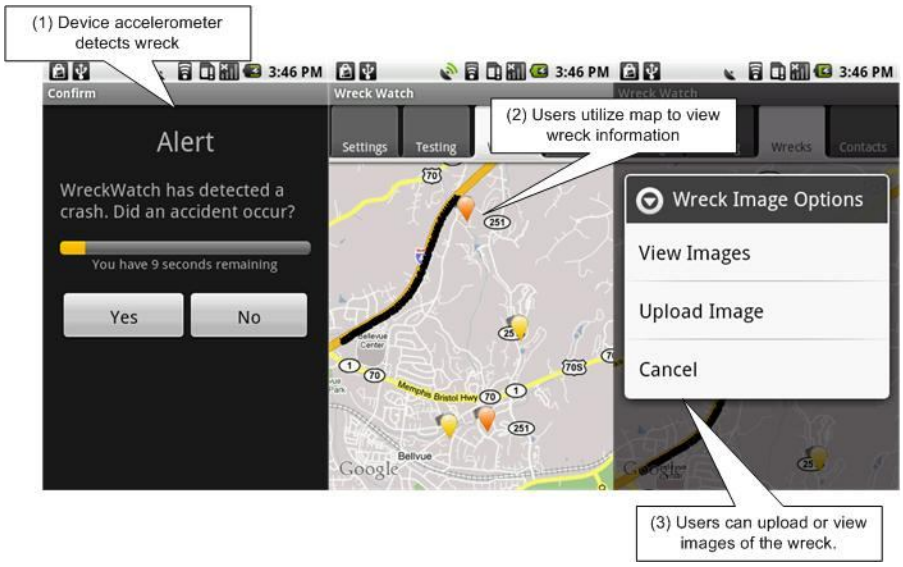

Fig. 1. Wreck Watch Behavior

Wreck Watch runs on Google Android smartphones to detect car accidents (1) by analyzing data from the device's GPS receiver and accelerometer and looking for sudden acceleration events from a high velocity indicative of a collision. Car accident data is then posted to an HTTP server where (2) it can be retrieved by other devices in the area to help alleviate traffic congestion, notify first responders, and (3) provide accident photos to an emergency response center. Users of Wreck Watch can also elect to have certain people contacted in the event of an accident via SMS message or a digital PBX. Figure 1 shows this behavior of Wreck Watch.

Since the Wreck Watch application runs continuously in the background, it must conserve its power consumption. The application needs to run at all times and consume a great deal of sensor information to accurately detect wrecks. If not designed properly, therefore, these characteristics could result in a substantial decrease in battery life. In our testing, for example, the Wreck Watch application was able to completely drain the device battery in less than an hour simply through its use of sensors and network connectivity.

In the case of Wi-Fi, the radio represents nearly 70% of device power consumption [2] and in extreme cases can consume 100 times the power of one CPU instruction to transmit one byte of data [3]. The amount of power consumed by the network adapter is generally proportional to the amount of information transmitted [1]. The framing and overhead associated with each protocol can therefore significantly affect the power consumption of the network adapter. Prior work [5] demonstrated that significant power savings could be achieved by modifying the MAC layer to minimize collisions and maximize time spent in the idle state. This work also recognized that network operations generally involved only the CPU and transceiver and by reducing client-side processing, they could substantially reduce power consumed by network transactions. Similarly, other work [7] demonstrated that such power savings could also be achieved through transport layer modifications.

Although MAC and transport layer modifications are typically beyond the scope of most software projects, especially mobile application development, the data transmitted on the network can be optimized so it is as lightweight as possible, thereby accomplishing, on a much smaller scale, some of the same effects. The remainder of this paper uses the Wreck Watch application to showcase key design challenges that developers face when building power-aware applications for mobile devices.

2.2 Design and Behavioral Challenges of Mobile Application Development

Despite the ease with which mobile applications can be developed via advanced SDKs (such as Google Android and Apple iPhone) developers still face many challenges related to power consumption. If developers do not fully understand the implications of their designs, they can substantially reduce device performance. Battery life represents a major metric used to compare devices and can be influenced significantly by design decisions. Designing mobile applications while remaining cognizant of battery performance presents the following challenges to developers:

- ***Challenge 1: Accurately predicting battery consumption of arbitrary architectural decisions is hard.*** Each instruction executed can result in the consumption of an unknown amount of battery power. Accurately predicting the power consumed for each line of code is hard given the level of abstraction present in modern SDKs, as well as the complexity and numerous variations between physical devices. Moreover, disregarding language commonalities between completely unrelated devices, mobile platforms, such as Android, are designed to operate on a plethora of hardware configurations, which may affect the power consumption of a given configuration.
- ***Challenge 2: Trade-offs between performance and battery life are not readily apparent.*** Although performance and power consumption are generally design tradeoffs, the actual relationship between the two metrics is not readily apparent. For example, when comparing two networking protocols, plain HTTP and SOAP, plain HTTP might operate much faster requiring only 10 ms to transmit the data SOAP requires 50 ms to transmit. At the same time, HTTP might consume .5

mW, while SOAP consumes 1.5 mW. Without the context of real-world performance in a physical device it would be difficult to predict the overhead associated with SOAP. Moreover, this data may vary from one device to the next.

- ***Challenge 3: Effects of transmission medium on power consumed are largely device, application, and environment specific.*** Wireless radios consume a substantial amount of device power relative to other mobile-device components [6], where the power consumed is directly proportional to the amount of information transmitted [1]. Each radio also provides differing data rates, as well as power consumption characteristics. Depending on the application, developers must choose the connection medium best suited to application requirements, such as medium availability and transmission rate. The differences between transmission media are generally subtle and may even depend on environmental factors [10], such as network congestion that are impossible to accurately predict. To deterministically and accurately quantify performance, therefore, testing must be performed in environmentally-accurate situations.

- ***Challenge 4: It is hard to accurately predict the effects of reducing sensor data consumption rates on power utilization.*** To provide the most accurate readings and results, device sensors would be polled as frequently as they sample data. This method consumes the most possible power, however, by not only requiring that the sensor be enabled constantly, but by also increasing the amount of data the device must process. In turn, reducing the time that the sensor is active significantly reduces the effectiveness and accuracy of the readings. Determining the exact amount of power saved by a reduction in polling rate or other sensor accuracy change is difficult without profiling such a change on a device.

- ***Challenge 5: Accurately assessing effects of different communication protocols on performance is hard without real-world analysis.*** Each communication protocol has a specific overhead associated with it that directly affects its overall throughput. The natural choice would be to select the protocol with the lowest overhead. While this decision yields the highest performance, it also results in a tightly coupled architecture [11] and substantially increases production time. That protocol would only be useful for the specific data set for which it was designed, in contrast to a standardized protocol, such as HTTP. Standardized protocols often support features that are unnecessary for many mobile applications, however, making the additional data required for HTTP transactions completely useless. It is challenging to predict how much of a tradeoff in performance is required to select the more extensible protocol because the power cost of such protocols cannot be known without profiling them in a real-world usage scenario.

Discussions on performance optimization have often focused on hardware- or firmware-level changes and ignored potential application layer enhancements [3] [5] [6]. Interestingly, this corresponds to the level of abstraction present in each layer: device drivers and hardware have little or no abstraction while software applications are often more thoroughly abstracted. It is this level of abstraction, however, that makes such optimizations challenging because often the developer has little or no control over the final machine code. Application code thus cannot be benchmarked until it has been fully developed and compiled.

Moreover, problems identified after the code is developed are substantially more costly to correct than those that can be identified at design time. The value of optimizing the performance of an application before any code is written is therefore of great value. Moreover, because power consumption is generally hardware-specific [1] such optimizations result in a tightly coupled architecture that requires the developer to rewrite code to benchmark other configurations.

3 Model-Based Testing and Performance Analysis

This section describes our current work in developing a modeling language extension to the *Generic Eclipse Modeling System* (GEMS) (www.eclipse.org/gmt/gems) [17], called the *System Power Optimization Tool* (SPOT) for optimizing performance and power consumption of mobile applications at design time. GEMS is an MDE tool for building Domain Specific Modeling Languages (DSMLs) for the Eclipse platform. The goal of SPOT is to allow developers to rapidly model potential application architectures and obtain feedback on the performance and power consumption of the architecture without manual implementation. The performance data is produced by generating instrumented architectural emulation code from the architectural model that is then run on the target hardware. After execution, cumulative results can be downloaded from the target device for analysis. This section describes the modeling language, emulation code generation, and performance measurement infrastructure that we are developing to address the five challenges described in Section 2.2.

3.1 Mobile Application Architecture Modeling and Power Consumption Estimation with SPOT

To accurately model mobile device applications, SPOT provides a domain-specific modeling language (DSML) with components that (1) represent key, resource-consuming aspects of a mobile application's architecture and (2) allows developers to specify visual diagrams of a mobile application architecture, as shown in the workflow diagram in Figure 2. SPOT's DSML, called the *System Power Optimization Modeling Language* (SPOML), allows developers to build architectural specifications from the following types of model elements:

- **CPU consumers**, which represent computationally intense code-segments such as location-based notifications that require distance calculations on hundreds of points.
- **Memory consumers**, which represent sections of application code that will incur heavy memory operations reducing performance and increasing power consumption, e.g., displaying an image, stored on disk, on the screen, etc.
- **Sensor data consumers**, which will poll device sensors at user-defined intervals.
- **Network consumers**, which periodically utilize network resources emulating actual application traffic

- **Screen drawing agents**, which interact with device graphics libraries, such as OpenGL, to consume power by rendering images to the display.

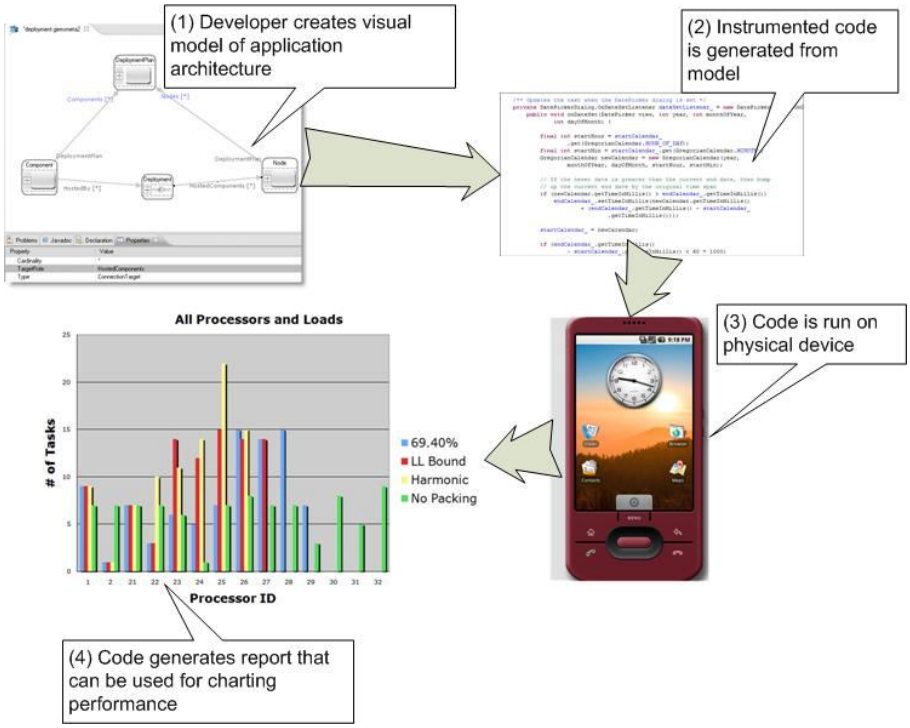

Fig. 2. SPOT Analysis Cycle

The sensor and network data consumers operate independently of application logic and simply present an interface through which their data can be accessed. The CPU consumer, however, need to incorporate application-specific logic, as well as logic from other aspects of the application. The CPU consumer module also allows for developers to integrate actual logic application as it becomes available to replace emulation code that is generated by SPOML.

To provide the software developer with the most flexibility and extensibility possible, SPOML provides them with many key power consumptive architectural options that would be present if they were actually writing device code. For example, if the device presents 10 possible options for granularity of GPS readings, SPOML provides all 10 possibilities via visual elements, such as drop down menus and check boxes. SPOML also provides for constraint checking that warns developers at design time if certain configuration options are unlikely to work together. Ultimately, SPOT provides developers with the ability to modify design characteristics rapidly and model their system without any application-specific logic, as well as provides them with a means to incorporate actual application code.

3.2 Architectural Emulation Code Generation

Due to the difficulty of estimating power consumption for an arbitrary device and software architecture it is essential to evaluate application performance on the actual physical hardware in production conditions. To accomplish this task, SPOT can automatically generate instrumented code to perform the functions outlined by the architecture modeled in SPOML. This code generation is done by traversing the in-memory object graph of the model and outputting optimized code to perform the resource-intensive operations specified in the model.

The architectural emulation code is constructed from several basic building blocks, as described above. The sensor consumers remain largely the same between applications and require little input from the user developing the model. The only variable in their construction is the rate at which they poll the sensor. They present an interface through which their data can be accessed.

The network consumer itself consists of several modules: a protocol, a transmission scheme and a payload interface. The payload interface defines methods that allow other components of the application to utilize the network connection and, for the purposes of emulation and analysis, this interface also helps define the structure of the data to transmit. The protocol module allows the developer to select from a set of pre-defined protocols (e.g., HTTP or SOAP) or create a custom protocol with a small amount of code. The transmission scheme defines a set of behaviors for how to transmit data back to the server, which allows developers to specify whether the application should transmit as soon as data is available, wait until a certain amount of data is available, or even wait until a certain connection medium is available (such as Wi-Fi or EDGE). Finally, the screen rendering agent allows users to specify the interval at which the screen is refreshed or invalidated for a given view.

Each module described above relies almost entirely on prewritten and optimized code. Of greater complexity for users are the CPU and memory consumers. Users may elect to utilize prewritten code that closely resembles the functionality they wish to provide. Alternatively, they can write their own code to use in these modules profile their architecture more accurately. This iterative approach allows developers to quickly model their architecture without writing detailed application logic and then as this code becomes available, refine their analysis to better represent the performance and behavior of the ultimate system.

3.3 Performance and Resource Consumption Management

When generating emulation code, SPOT also generates instrumentation code to record device performance and track power consumption. This code writes these metrics to a file on the device that can later be downloaded to a host machine for analysis after testing. This approach allows developers to quantitatively compare metrics such as application responsiveness (by way of processor idle time, etc), network utilization and throughput and battery longevity. These comparisons provide the developer with a means to quickly and accurately design a system that minimizes power consumption without sacrificing performance. In some instances, this analysis could even highlight

simple changes such as reducing the size of XML tags to reduce the overhead associated with downloading information from a server.

In each challenge presented in Section 2.2 we establish that through current methods, certain characteristics of a design can only be fully understood post-implementation. Additionally, with newer platforms such as Google's Android, the mobile device has become an embedded multi-application system. Since each device has significantly less resources than their tethered brethren, however, individual applications must be cognizant of their resource consumption. The value of understanding a given application's power consumption profile is thus greatly increased.

The solutions to each of these challenges lie within the same space: utilization of a model that can be used to accurately assess battery life. SPOT addresses mobile application performance analysis through the use of auto-generated code specified by a DSML, which allows users to estimate performance and power consumption early in the development process. Moreover, developers can perform continuous integration testing by replacing faux code with application logic as it is developed.

4 Concluding Remarks

The capabilities of mobile devices have increased substantially over the last several years and with platforms, such as Apple's iPhone and Google's Android, will no doubt continue to expand. These platforms have ushered in a new era of applications and have presented developers with a wealth of new opportunities. Unfortunately, with these new opportunities have come new challenges that developers must overcome to make the most of these cutting-edge platforms. In particular, predicting performance characteristics of a given design is hard, especially those characteristics associated with power consumption.

A promising approach to address these challenges is to enhance model-driven engineering (MDE) tools to enable developers to quickly understand the consequences of architectural decisions. These conclusions can be drawn long before implementation, significantly reducing production costs and time while substantially increasing battery longevity and overall system performance. From our experience developing SPOT, we have learned the following lessons:

- By utilizing MDE it becomes possible to quantitatively compare design decisions and deliver some level of optimization with regards to power consumption,
- Developing applications for platforms such as Android require extensive testing as hardware configurations can greatly influence performance, and
- It is impossible to completely profile a system configuration because ultimate device performance and power consumption depends on user interaction, network traffic and other applications on the device.

The WreckWatch application is available under the Apache open-source license and can be downloaded at http://vuphone.googlecode.com.

References

1. Feeney, L. & Nilsson, M., "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," *IEEE INFOCOM,* 2001*, 3*, 1548-1557.
2. Liu, T.; Sadler, C.; Zhang, P. & Martonosi, M., "Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet" *Proceedings of the 2nd international conference on Mobile systems, applications, and services,* 2004, 256-269.
3. Pering, T.; Agarwal, Y.; Gupta, R. & Want, R., "Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces," *Proceedings of the Annual ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys),* 2006.
4. Poole, J., "Model-driven architecture: Vision, standards and emerging technologies," *Workshop on Metamodeling and Adaptive Object Models, ECOOP,* 2001.
5. Chen, J.; Sivalingam, K.; Agrawal, P. & Kishore, S., "A comparison of mac protocols for wireless local networks based on battery power consumption," *IEEE INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies.*
6. Krashinsky, R. & Balakrishnan, H., "Minimizing energy for wireless web access with bounded slowdown," *Wireless Networks, Springer,* 2005*, 11*, 135-148.
7. Kravets, R. & Krishnan, P., "Application-driven power management for mobile communication," *Wireless Networks, Springer,* 2000*, 6*, 263-277.
8. Paunov, S.; Hill, J.; Schmidt, D.; Baker, S. & Slaby, J., "Domain-specific modeling languages for configuring and evaluating enterprise DRE system quality of service," *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on,* 2006.
9. Hill, J.; Tambe, S. & Gokhale, A., "Model-driven engineering for development-time QoS validation of component-based software systems," *Proceeding of International Conference on Engineering of Component Based Systems,* 2007.
10. Carvalho, M.; Margi, C.; Obraczka, K. & others, "Garcia-Luna-Aceves. Modeling energy consumption in single-hop IEEE 802.11 ad hoc networks," *Thirteenth International Conference on Computer Communications and Networks (ICCCN'04,* 2004, 367-37.
11. Gay, D.; Levis, P. & Culler, D. "Software design patterns for TinyOS" *ACM New York, NY, USA,* 2007.
12. Boehm, B. "A spiral model of software development and enhancement" Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research, Wiley-IEEE Computer Society Pr, 2007, 21, 345.
13. Tan, E.; Guo, L.; Chen, S. & Zhang, X. "PSM-throttling: Minimizing energy consumption for bulk data communications in WLANs" *IEEE International Conference on Network Protocols, 2007. ICNP 2007,* 2007, 123-132.
14. Kang, J.; Park, C.; Seo, S.; Choi, M. & Hong, J. "User-Centric Prediction for Battery Lifetime of Mobile Devices" Proceedings of the 11th Asia-Pacific Symposium on

Network Operations and Management: Challenges for Next Generation Network Operations and Service Management, 2008, 531-534.

15. Kent, S. "Model Driven Engineering" *Lecture notes in computer science, Springer,* 2002, 286-298.
16. Lédeczi, &A. Bakay, A.; Maroti, M.; Völgyesi, P.; Nordstrom, G.; Sprinkle, J. & Karsai, G. "Composing domain-specific design environments" *Computer, IEEE Computer Society,* 2001, 44-51.
17. Jules White, Douglas C. Schmidt, Sean Mulligan, "The Generic Eclipse Modeling System", Model-Driven Development Tool Implementer's Forum at the 45th International Conference on Objects, Models, Components and Patterns, June, 2007, Zurich Switzerland.