# Architecture Patterns for Mobile Systems in Resource-Constrained Environments

Grace A. Lewis, Soumya Simanta, Marc Novakouski, Gene Cahill, Jeff Boleng, Edwin Morris, James Root
Carnegie Mellon Software Engineering Institute (SEI)
Advanced Mobile Systems Initiative
Pittsburgh, PA USA
{glewis, ssimanta, novakom, gmcahill, jlboleng, ejm, jdroot}@sei.cmu.edu

***Abstract*— Soldiers, first responders and other personnel operating at the tactical edge increasingly make use of mobile devices to help with tasks such as face recognition, language translation, decision-making and mission planning. Tactical-edge environments are characterized by limited resources, dynamic context, high stress and poor connectivity. This paper focuses on three architecture patterns that address these conditions. The *Data Source Integration* pattern uses server-side standardized definitions of live or cached geo-located data feeds that can be customized and filtered on a single, map-based user interface on a mobile device. The *Group Context Awareness* pattern uses context obtained from groups of handheld devices operating as part of a team to make sure that the right information is displayed to the right soldier at the right time. The *Cloudlet-Based Cyber-Foraging* pattern uses cloudlets as code-offload elements to optimize resources and increase computation power of mobile devices. Cloudlets are discoverable, localized, stateless servers running one or more virtual machines on which users can offload resource-intensive computations from their mobile devices. Prototype applications have been implemented for each of these patterns. Experiment results and participation in exercises have shown the effectiveness of the patterns in addressing the challenges of resource-constrained environments.**

***Keywords—mobile computing, mobile systems, architecture patterns, software architecture, resource-constrained environments, tactical edge***

## I. INTRODUCTION

Soldiers, first responders and other personnel operating at the tactical edge increasingly make use of mobile devices to help with tasks such as face recognition, language translation, decision-making and mission planning. These resource-constrained, edge environments are characterized by

- occasional to frequent dismounted operations
- intermittent or no connectivity to traditional infrastructure;
- a fast paced, highly fluid and unpredictable environment;
- the potential of large amounts of raw data and information;
- resource challenges (i.e. power, computing, etc.); and
- periods of very high stress and cognitive load

Given these characteristics mobile systems deployed in these environments must address the following challenges to meet mission needs:

- Provide situational awareness and data analysis for soldiers disconnected from high-speed data reach back and only able to depend on tactical radio data networks
- Reduce cognitive load and complexity for dismounted soldiers, particularly in situations of high stress
- Increase computing power, data access, and survivability of computing capabilities in tactical environments while reducing demands on person-carried resources

This paper presents an initial set of patterns for mobile systems in resource-constrained environments that address these challenges. Section II presents related work. Section III describes the Data Source Integration pattern that provides access to disparate and potentially unknown data sources while addressing the challenges of limited resources, high cognitive load, information overload and dynamic environments. Section IV describes the Group Context awareness pattern that enabled teams to share and make decisions that consider group context while addressing the challenges of limited resources, uncertainty of available infrastructure, and high cognitive load. Section V describes the Cloudlet-Based Cyber-Foraging pattern that increases the computing power or mobile devices while addressing the challenges of limited processing power and battery life, unreliable networks, uncertainty of available infrastructure, and uncertainty of connectivity to the enterprise. Section VI describes our current and future work in this area. Section VII concludes the paper.

## II. RELATED WORK

In software engineering, a pattern represents a proven solution to a recurring design problem. Gamma et al were the first to develop design patterns for object-oriented systems [1]. These design patterns were manifested as low-level reusable object-oriented designs described using a standard template. Buschmann et al were the first to present architecture patterns for large-scale applications that could be composed to promote certain system qualities [2]. Our patterns are described using an abbreviated version of their template. Since then, architecture patterns have been developed for specific types of applications or domains, such as concurrent and networked systems [3], enterprise applications [4], enterprise integration [5], resource management [6], and distributed systems [7]. Specific to mobile systems, patterns have been presented for mobile web information systems focusing on web access and customization [8] and service-based mobile applications focusing on code offload [9]. To the best of our knowledge, we are the first to explore architecture patterns beyond code offloading for mobile systems in resource-constrained environments.

## III. The Data Source Integration Pattern

### A. Motivation

Map-based apps with optional data layers or overlays, such as Google Maps, can provide SA to edge users, but only if the apps provide access to a far greater range and type of mission-oriented data. To support the dynamic aspect of edge environments and missions, SA applications in the field must support the rapid creation of mashups that consolidate information and support concurrent visualization of information from multiple sources that might not be known in advance.

### B. Problem

SA capabilities at the edge require a mashup capability that allows end users to connect to and display data from multiple unrelated data sources that, when combined, provide a more complete situational presentation than the individual sources alone. To address the challenges of resource-constrained environments, required capabilities of edge-enabled SA solutions include rapid incorporation of new data sources; minimized information overload; user control of data sources, data volume and visualization method; and simple use.

### C. Solution

We propose a solution for data source integration that supports the rapid creation of mashups of geo-located data from multiple sources and displays that data on handheld devices to support situational awareness. A reference architecture is presented in Figure 1 and a specific implementation is documented in [10].

The main elements of the architecture are the *Client App* on the *Mobile Device* and the *Data Source Manager* on the *Data Integration Server*. The client app is composed of a set of *Views* that most likely contains a map-based view to display the retrieved geo-located data and a search view to enable end users to create data filters. A key element of the client app is the *Rule Builder*, which translates end-user created filters into rules that are sent as part of the Data Request from the *Data Client* to the *Data Server.*

On the server side, each *Data Source* is separately described by a *Data Source Descriptor*. Data sources can be local (i.e., on the same server), on a server in the same local network, or in the cloud. Upon receiving the data request from the client, the data server sends the request to the *Data Connector and Filter* that corresponds to the requested data source. This component contains the data access logic and the rules engine that executes the filtering rules that are contained within the data request.

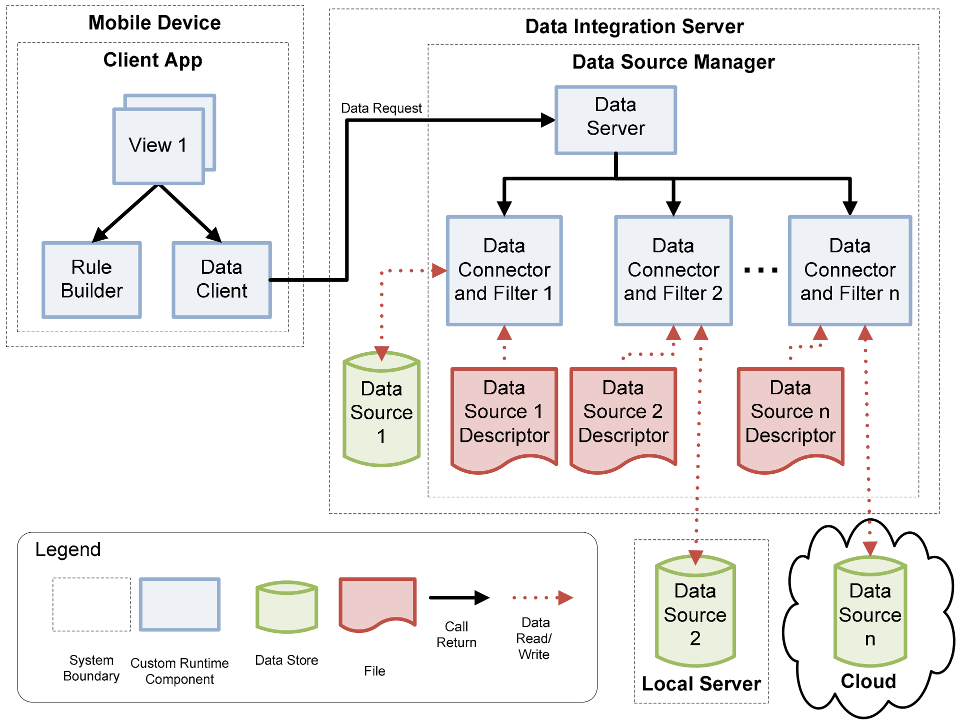

**Figure 1. Reference Architecture for Rapid Integration of Situational Awareness Data**

### D. *Impact*

The Data Source Integration pattern addresses the challenges of dynamic environments, high cognitive load, information overload, and limited resources. Server-side, separately defined data sources simplify the addition and removal of data sources without affecting existing data models or clients. In addition, because the differences in data sources are addressed at the server side, it enables the creation of a unified user interface (as opposed to a user interface per data source) that reduces cognitive load.

User-defined filters address the challenge of cognitive load and information overload because users at runtime can determine how much and what information to see. Data is filtered on the server before sending it to the mobile client, which reduces bandwidth utilization. However, the data formats used in the data requests will also have an effect on bandwidth utilization (e.g., bandwidth-hungry formats such as XML and SOAP vs. more compact data formats such as protocol buffers). Server-side data filtering also addresses resource limitations because potentially computation-intensive filtering rules are offloaded and executed on the server and not on the mobile device (see Section V).

Finally, while the link from the mobile client to the data integration server will likely be relatively low-bandwidth wireless, the link from the server to the data sources can be a higher-bandwidth wireless or, potentially, a wired link. The faster data rates and higher bandwidth of the links can result in lower overall latency to the mobile clients.

### E. *Variants*

The Data Source Integration pattern can be instantiated in multiple ways by using different combinations of technologies. Even though the solution presented in Figure 1 is based on a request-response interaction paradigm, publish-subscribe would also be possible. In this case the equivalent of the data request would be a data source subscription that contains rules describing the specifics regarding data fields and volume to be published. In addition, the presented solution uses location as the "mashup field" because it is a common field in available SA information. The use of other mashup fields is possible and would have to be defined and used by specific views in the client app. Finally, depending on needs and server capabilities, all results of data requests could be cached. This strategy takes advantage of the fact that individuals working in the same area often make repeated accesses to the same data. A more predictive caching strategy based on context information such as location could also support disconnected operation.

## IV. The Group Context Awareness Pattern

### A. *Motivation*

Edge environments demand that individuals operate cooperatively in teams. When teams operate in high stress environments such as search and rescue operations, or tactical combat engagements, it is crucial that every team member is able to execute their assigned tasks to ensure the teams' larger mission is accomplished, and that all the members of the team are provided the right information at the right time in order to ensure their safety and effective actions.

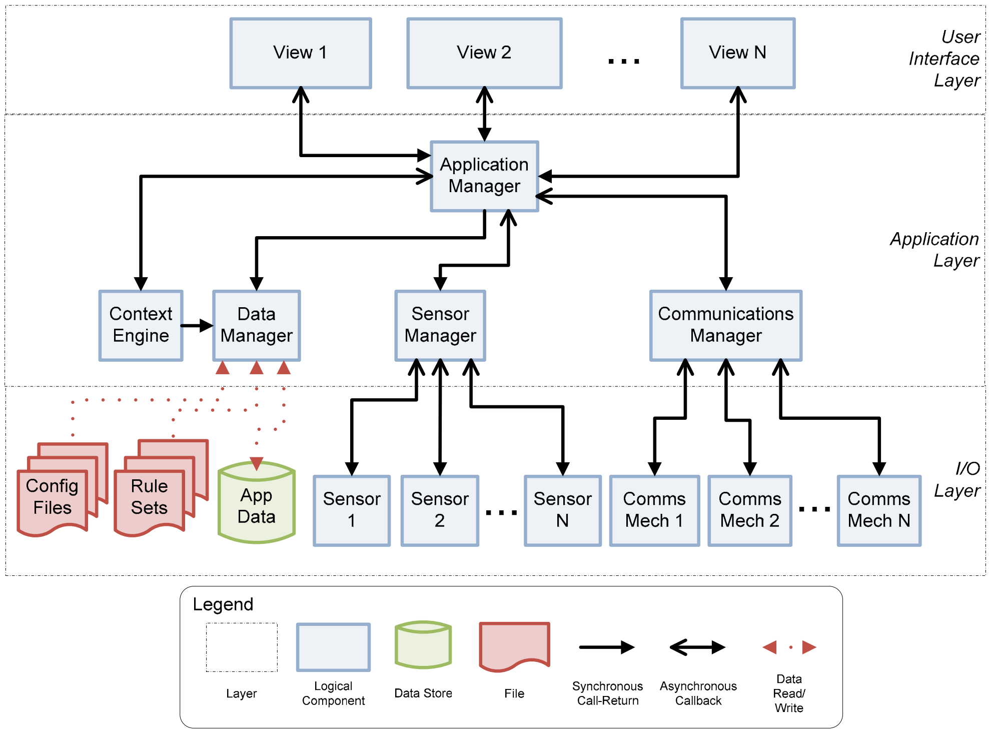

**Figure 2. Reference Architecture for Group-Context-Aware Mobile Applications**

### B. Problem

An architecture for group context awareness must be flexible enough to create, analyze, and react to unpredictable events that were never imagined during planning. Similarly, the architecture must be resilient to intermittent communications connectivity and opportunistic in using communication capabilities as they become available. The architecture should also provide the capability to manage resources on the mobile device such that resources are used as efficiently as possible to maximize the availability of the system. Finally, the architecture must address cognitive load by providing mechanisms for supporting the information needs of the group while requiring minimal attention and interaction.

### C. Solution

We propose a solution for group context awareness that addresses flexibility, resource management, and cognitive load. A reference architecture for group-context-aware mobile systems is presented in Figure 2 and documented in [11].

The architecture follows the basic architecture for context-aware mobile applications proposed in [12] that divides the architecture into context capture, context reasoning and aggregation and context visualization. This architecture also follows the common model-view-controller (MVC) pattern. The model is the *App Data* in the *I/O Layer* which is managed by the *Data Manager*. The controller is the *Application Manager* in conjunction with the other components in *Application Layer*. The view is the set of *Views* in the *User Interface Layer*. These views register an interest in events produced by the system and display data accordingly. The views can also input context data from the user.

The *Context Engine* is at the core of the architecture and is the central processor for all context information used by the application. As device sensors report new data and context data is received from group members, it is passed through the engine so that new events are detected as they occur. Context data is processed based on mission-specific *Rule Sets* that can be created, added, and swapped as needed. The *Sensor Manager* accepts data from *Sensors* on the mobile device, such as position sensors, movement sensors, light and proximity sensors, etc. It is highly-extensible to incorporate new sensors easily. The *Communications Manager* acts as the gateway for all external communications. It is also highly extensible to support multiple *Communication Mechanisms*.

### D. Impact

Group Context Awareness addresses the challenges of limited battery life, uncertainty of available infrastructure, and high cognitive load. The layered architecture/MVC approach allows for maximum extensibility and modifiability. In addition, the XML-based configuration approach and group context model (discussed further in [11]) allow for maximum flexibility in defining and creating arbitrary events and system responses for a given mission.

Flexibility in managing sensors and communications is provided by the manager approach. Combining generalized control of sensors and communications with the ability of the context engine to react to different situations allows a system implementing this pattern to tailor sensor sensitivity and reporting rate to the situation, as well as to choose when and how to communicate to other groups.

Finally, the key advantage of a group context model and context engine is management of cognitive load. Understanding the situation of the individual and the group allows the context engine to intelligently manage both when and how information is presented to the user. This results in a system that knows when to do nothing, when to queue information for later presentation, and when to interrupt the user. The MVC pattern also supports this goal by supporting the creation of arbitrary views, allowing the tailoring of information presentation to a given domain or context.

### E. Variants

The power of the Group Context Awareness pattern is its flexibility. The presented reference architecture provides the capability for communications, sensor polling, event creation and reaction through the context engine, and customized display of data through arbitrary UI views. This pattern could be used to instantiate applications that use any or all of these components. More interestingly, the pattern could potentially be used to implement machine-learning-like behavior, by gathering data and analyzing it over some period of time, then using the context engine to react to events identified by the machine learning algorithms.

## V. The Cloudlet-Based Cyber-Foraging pattern

### A. Motivation

Applications that are useful in resource-constrained, mission execution environments include, among others, face recognition, natural language processing, and language translation. These resource-intensive applications require great amounts of battery and computing power and might not even be able to execute on the mobile device due to the complexity of the algorithms required.

### B. Problem

Cyber foraging, as first introduced by Satyanarayanan [13], is a technique to enable resource-poor mobile devices to leverage external computing power. A mobile device offloads code to a surrogate, taking advantage of a more powerful hardware infrastructure. This surrogate executes the code and returns the computational result to its client. However, most existing cyber-foraging solutions assume connectivity to the cloud or tightly couple offloaded applications to a specific infrastructure, which make them infeasible in resource-constrained environments or harder to provision.

### C. Solution

We propose a solution for cyber-foraging based on cloudlets — discoverable, generic, stateless servers located in single-hop proximity of mobile devices, that can operate in disconnected mode and are virtual-machine (VM) based to promote flexibility and mobility [14]. A reference architecture for cloudlet-based cyber-foraging is presented in Figure 3 and documented in [15].

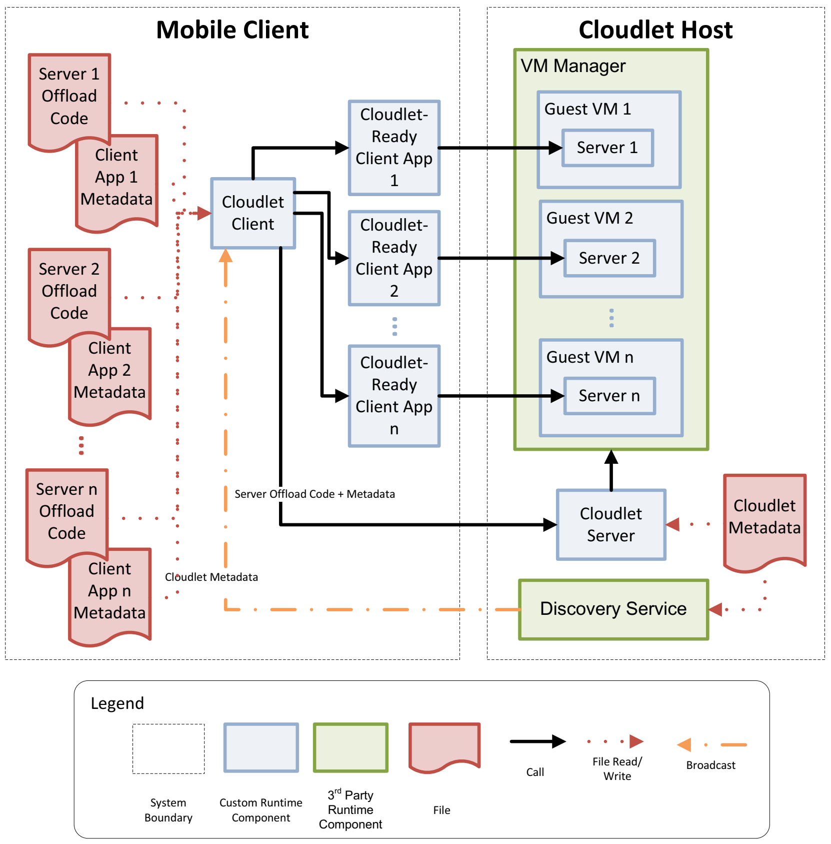

**Figure 3. Reference Architecture for Cloudlet-Based Cyber-Foraging**

The main elements of the architecture are the *Mobile Client* and the *Cloudlet Host*. A *Discovery Service* running inside the cloudlet host publishes *Cloudlet Metadata* that is used by the *Cloudlet Client* to determine the appropriate cloudlet for offload and to connect to the cloudlet. Metadata can range from a simple IP address and port to connect to the cloudlet server to complex data structures describing cloudlet capabilities. Every application is composed of a *Cloudlet-Ready Client App* that corresponds to the client portion, the *Server Offload Code* that corresponds to the server portion, and the *Client App Metadata* that contains information that is used by the cloudlet client and the cloudlet server to negotiate and carry out the code offload process. Once a cloudlet is identified for offload, the cloudlet client sends the server offload code and client app metadata to the *Cloudlet Server*. The cloudlet server then deploys the server code inside a *Guest VM* inside the *VM Manager*. The server offload code can range from source code, to application packages, to complete VMs. Once the deployment is complete, the cloudlet server is notified that the server is ready for execution and the client app is launched.

## D. *Impact*

Cloudlet-based cyber-foraging addresses the challenges of limited processing power and battery life by offloading expensive computation to more powerful cloudlets. Because cloudlets are stateless — they can operate in disconnected mode and only have to be connected to the cloud for provisioning — they address the challenge of uncertainty of connectivity to the enterprise. The challenge of uncertainty of available infrastructure and unreliable networks is addressed because offloaded code is deployed in a basic VM Manager such as KVM or VMWare. Migrating offloaded code to a different cloud can be done via live VM migration. This also increases the survivability of the solution.

However, Simanta [15] and Messinger [18] both demonstrate that the size of the server offload code is directly proportional to energy consumption and application-ready time. While VM synthesis, which is the cloudlet provisioning mechanism used in [15], provides great flexibility and places minimal requirements on the infrastructure, the size of the server offload code is five time larger than with application virtualization which is the technique used in [18]. The limitation of application virtualization however is that correct execution is only guaranteed if all application dependencies are captured, which is not a problem with VM synthesis. This means that mobile systems that implement this pattern have to balance server offload code size with desired system qualities and infrastructure requirements.

## E. *Variants*

Even though in traditional cyber-foraging solutions code is always offloaded from the mobile device to the surrogate (cloudlet), a variant to this pattern is to offload the client app from the cloudlet to the client. Cloudlet-based cyber-foraging is an “always offload” strategy, which means that server code is always offloaded to the surrogate. Variants of this pattern include static partitioning strategies such as MAUI [16] or dynamic partitioning strategies such as CloneCloud [17] to determine whether it is more energy efficient to execute locally or remotely. However, this variant assumes that the code can execute locally or remotely.

## VI. Current and Future Work

The architecture patterns presented in this paper have been implemented for Android using current, open source technologies to demonstrate the validity of the patterns as well as their technical feasibility.

- The *Data Source Integration* pattern has been instantiated to support the rapid integration of DoD and public data feeds onto a map-based user interface that can be easily configured by the end user [10].
- The *Group-Context Awareness* pattern has been instantiated to support real-time sharing of SA information enhanced with contextual information using small tablets and communicating peer-to-peer via radios [11].
- The Cloudlet-Based Cyber-Foraging pattern has been implemented using two different provisioning strategies — VM synthesis [15] and application virtualization [18]. We are in the process of implementing two additional strategies that will minimize the communication between the mobile device and the cloudlet for provisioning.

These implementations have been taken to field experiments and exercises to demonstrate their operational feasibility. Consistent with the SEI’s mission, we are in the process of finding transition partners for which the patterns can be instantiated according to specific missions and organizational requirements and moved into real systems.

In addition, as stated in Fowler [4], patterns are combined to create larger-scale solutions. We are currently working on the combination of these patterns to support context-aware, rapidly deployed SA solutions. The Data Integration Server is rapidly deployed onto a cloudlet using Cloudlet-Based Cyber-Foraging. Group-Context-Awareness is used to tailor the information that is retrieved and visualized on each mobile device. We are also working on additional architecture patterns to account for dynamic environments.

## VII. Conclusions

We have presented a set of architecture patterns for mobile systems in resource-constrained environments that support first responders and military personnel operating in edge environments. These architecture patterns are driven by flexibility, resource efficiency, and usability, which are key quality attributes for systems at the tactical edge. The goal of these patterns is to enable system architects to instantiate them using a variety of technologies that can meet functional and quality requirements.

## Acknowledgment

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution (DM-0000316).

## References

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison-Wesley, Boston, MA, 1995.

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, Inc., New York, NY, USA, 1996

[3] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects, Wiley, 2000.

[4] Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] Gregor Hohpe and Bobby Woolf. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, 1st Edition. Addison-Wesley Professional, 2003.

[6] Michael Kircher and Prashant Jain. Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management. Wiley, 2004.

[7] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing. Wiley, 2007.

[8] Walter A. Risi and Gustavo Rossi. An architectural pattern catalogue for mobile web information systems. Int. J. Mob. Commun. 2, 3 (September 2004), 235-247.

[9] Jennifer Kim. Architectural patterns for service-based mobile applications," Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on , vol., no., pp.1,4, 13-15 Dec. 2010.

[10] Soumya Simanta, Gene Cahill, and Edwin Morris. eMontage: An Architecture for Rapid Integration of Situational Awareness Data at the Edge. Engineering of Mobile-Enabled Systems (MOBS), 2013 ICSE Workshop on, May 2013 (to be published in September 2013).

[11] Grace Lewis, Marc Novakouski, and Enrique Sánchez. A Reference Architecture for Group-Context-Aware Mobile Applications. In Mobile Computing, Applications, and Services (pp. 44-63). Springer Berlin Heidelberg, 2013.

[12] Anind Dey. Understanding and Using Context, Springer-Verlag London Ltd. Personal and Ubiquitous Computing 5:4—7, 2001.

[13] Mahadev Satyanarayanan. Pervasive Computing: Vision and Challenges. IEEE Personal Communications, pp. 10-17, 2001.

[14] Mahadev Satyanarayanan; Paramvir Bahl, Ramón Cáceres, and Nigel Davies. The Case for VM-Based Cloudlets in Mobile Computing, Pervasive Computing, IEEE , vol.8, no.4, pp.14,23, Oct.-Dec. 2009.

[15] Soumya Simanta, Grace A. Lewis, Edwin Morris, Kiryong Ha, and Mahadev Satyanarayanan. A Reference Architecture for Mobile Code Offload in Hostile Environments, Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on , vol., no., pp.282,286, 20-24 Aug. 2012.

[16] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. In Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10). ACM, New York, NY, USA, 49-62. 2010.

[17] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: elastic execution between mobile device and cloud. In Proceedings of the sixth conference on Computer systems (EuroSys '11). ACM, New York, NY, USA, 301-314. 2011.

[18] Dominik Messinger and Grace Lewis. Application Virtualization as a Strategy for Cyber Foraging in Resource-Constrained Environments (CMU/SEI-2013-TN-007). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2013.