A Reference Architecture for Group-Context-Aware Mobile Applications

Grace Lewis, Marc Novakouski, and Enrique Sánchez

CMU Software Engineering Institute
4500 Fifth Ave. Pittsburgh, PA 15213 USA
{glewis,novakom,eysanchez}@sei.cmu.edu

Abstract. Handheld mobile technology is reaching first responders and soldiers in the field to help with mission execution. A characteristic of mission execution environments is that people are typically deployed in teams or groups to execute the mission. Most commercially-available context-aware mobile applications are based on context expressed mainly as location and time of an individual device plus the device user's preferences or history. This work extends context to consider the group that the individual is a part of and presents a reference architecture for group-context-aware mobile applications that integrates contextual information from individuals and nearby team members operating to execute a mission. The architecture is highly extensible to support changes in context data models, context data storage mechanisms, context reasoning engines and rules, sensors, communication mechanisms and context views. A prototype implementation was built to demonstrate the validity and extensibility of the reference architecture.

Keywords: context-awareness, mobile applications, Android, software architecture, reference architecture.

1 Introduction and Motivation

Handheld mobile technology is reaching first responders and soldiers in the field to help with mission execution. These individuals operate at the tactical edge, which is a term used to describe hostile environments with limited resources, from disaster relief areas in countries like Haiti and Japan, to war zones in Afghanistan.

A major challenge at the tactical edge is getting relevant information at the time it is needed. Causes include reliance on easily misplaced paper reports, one-way information flow (up the chain of command but not down), and the lack of network bandwidth and handheld devices to access information. Improved bandwidth and new devices can improve reporting and increase the volume of information, but these advances will also create information overload.

An important characteristic of mission execution environments is that people are typically deployed in teams or groups to execute the mission. For example, first responders in disaster areas cannot effectively pursue humanitarian tasks without coordination. Similarly, squads of warfighters in theater must coordinate very closely in order to accomplish missions, perform peacekeeping tasks, or even stay alive.

D. Uhler, K. Mehta, and J.L. Wong (Eds.): MobiCase 2012, LNICST 110, pp. 44–63, 2013.

Imagine a scenario where each first responder is given a mobile device with applications and information that will help them execute their mission. What will happen over time is that as first responders move away from the base, the information that is critical is determined by what the individuals and the group as a whole feel, see, hear, smell, or even by situations they cannot directly sense, such as high levels of radiation [1]. In addition, because of the type of work executed by first responders, especially in emergency situations, they are not in position to search for information, or scroll through multiple screens on a device to display the appropriate data when it is needed most. What they need is a capability that can sense as much of the emerging group context as possible, apply that context to share data with the group, and filter data such that only the most relevant information is shared and displayed.

The work presented in this paper is a reference architecture for group-context-aware mobile applications that enables the integration of contextual information from individuals, nearby team members, and potentially the enterprise to support a team executing a mission. Specific innovations of this work include consideration of a wide range of contextual information, including the dynamics of a group operating to achieve a common mission goal. In order to better interact with collaborators and quickly incorporate technological advances, the architecture is highly extensible to support changes in context data models, context data storage mechanisms, context reasoning engines and rules, sensors, communication mechanisms and context views. Section 2 introduces the concept of group context awareness as related to mobile applications at the tactical edge. Section 3 presents the reference architecture. Section 4 presents the architecture decisions and tradeoffs to support extensibility. Section 5 presents a prototype implementation for task management on the Android platform that validates the reference architecture. Section 6 provides a summary of related work. Finally, Section 7 presents conclusions and future work.

2 Group Context Awareness

There are many definitions of context related to context-aware applications [2][3][4][5][6][7][8][9][10][11][12]. Based on a synthesis of available definitions, we define context as any information that can be used to characterize an entity — person, place, or object — such as its properties, behavior, and surrounding environment. We define a context-aware application as an application that uses contextual information to modify its behavior, adapt its user interface, or filter data accordingly.

Most commercially-available context-aware mobile applications (apps) are based on context expressed mainly as location and time of an individual device plus the device user's preferences or history. For example, an app recommends a list of restaurants close to the current location of a user and orders them according to user cuisine preferences combined with the type of cuisine selected in the past by that user.

The guest editor introduction to a recent special issue on context-aware computing presents a challenge for context-aware system developers to work beyond search and location-based services to consider a larger set of context entities in order to improve

their value [13]. Consistent with this statement, the work presented in this paper extends context beyond location and time of an individual user to consider the context of the group that the individual is a part of (e.g. a rescue team). A group-context-aware mobile app first considers individual user context and then relates that information to the group context, thereby helping users understand both their own state as well as the state of the group in which they participate. Desired capabilities of group-context-aware mobile applications in hostile environments include

- Capture and store context information on a mobile device in a non-intrusive manner to reduce cognitive overload and without imposing an unreasonable burden on handheld device resources
- Disseminate context information to group members using whatever communications mechanisms are available at the moment
- Integrate local and group context information to improve mission effectiveness by only sharing and displaying information that is relevant to the individual and mission according to configurable rules

The following section presents a reference architecture for implementing group-context aware mobile applications that enables these capabilities.

3 Reference Architecture

The development of any software architecture should start with a definition of business drivers [14]. Given the early stages of the research project, as well as the fast speed at which technology is changing in the mobile space, we defined the following business drivers

1. Opportunistic integration of new technology
2. Ease of integration with components produced by collaborators
3. Applicability of architecture to different edge-enabled applications

To meet business drivers we defined extensibility as the main architectural driver, expressed as eight scenarios, as shown in Table 1. A sample scenario description, documented according to [14], is shown in Table 2.

Table 1. Extensibility scenarios

#	Name	Attribute Concern
1	Add a new sensor	Separation of concerns
2	Add a new sensor	Modifiability
3	Add a new communication mechanism	Separation of concerns
4	Add a new communication mechanism	Modifiability
5	Add a new content event/action	Separation of concerns
6	Add a new content event/action	Modifiability
7	Add a new context view	Separation of concerns
8	Add a new context view	Modifiability

Table 2. Scenario 3: Add a new communication mechanism

Scenario	Add a new communication mechanism	
Attribute	Extensibility	
Attribute concern	Separation of concerns	
Scenario refinement	**Stimulus**	Developer
	Stimulus source	Developer identifies a communication mechanism that can be used to share context data with other mobile devices
	Environment	Developer is sufficiently comfortable with application to make changes in a reasonable amount of time
	Artifact	Communications Manager of the context-aware application
	Response	Communications Manager is changed to implement message passing using the new communication mechanism
	Response measure	Aside from communication-mechanism-specific code, only the Communications Manager is changed to accommodate the new communications mechanism

The reference high-level architecture for group-context-aware mobile applications is a layered architecture as shown in Fig. 1. The architecture follows the basic architecture for context-aware mobile applications proposed in [2] that divides the architecture into context capture, context reasoning/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, the controller is the *Application Layer*, and the view is the *User Interface Layer*.

3.1 User Interface Layer

The *User Interface Layer* is the collection of views of context data. The views register an interest in events produced by the system and display data accordingly. The views can also input context data from the user.

3.2 Application Layer

The *Application Layer* is the core of the system. The components in this layer are responsible for managing context and creating events based on individual and group context.

The *Application Manager* is the central hub for all system activity.

The *Context Engine* 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, all data is passed through the engine so that new events are detected as they occur. Events are sent to the *Application Manager* for distribution to components that are interested in the events.

The *Sensor Manager* accepts data from sensors on the mobile device, such as position sensors, movement sensors, light and proximity sensors, etc. The *Sensor Manager* also controls sampling rate and change thresholds (the minimum variation in value to report a change) for each sensor.

The *Communications Manager* acts as the gateway for all external communications. Any messages to and from other devices are passed through the *Communications Manager*. It supports multiple communication mechanisms.

The *Data Manager* performs all CRUD (create, retrieve, update, delete) operations on context data and app data, and manages all access to the sensor configuration file and the context rule sets.

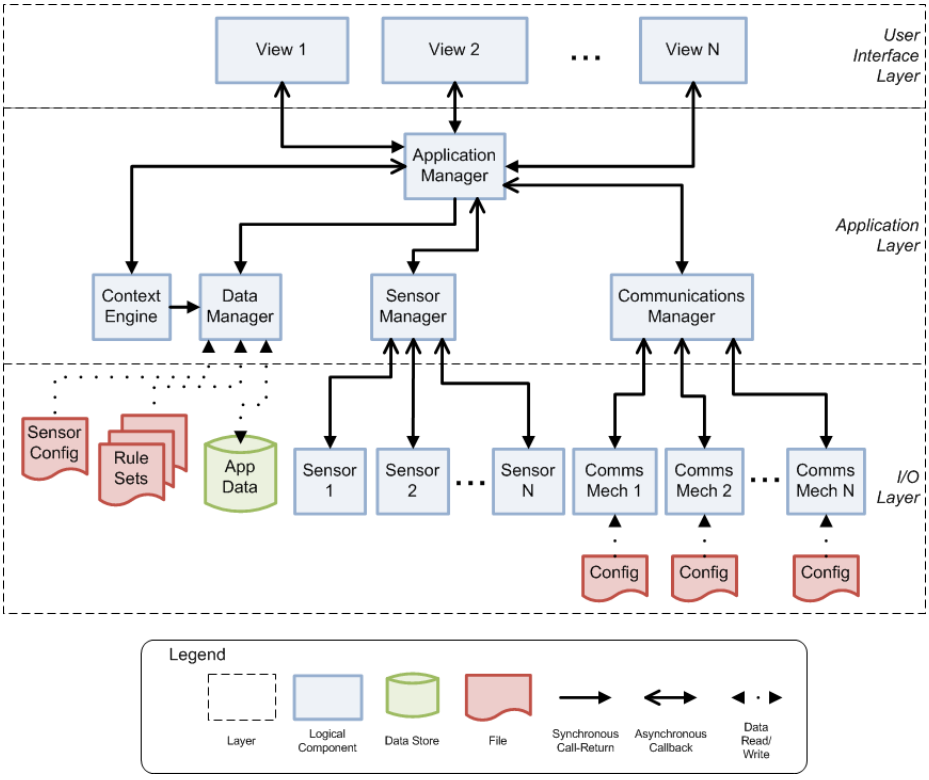

Fig. 1. Reference architecture for group-context-aware mobile applications

3.3 I/O Layer

The *I/O Layer* contains components that interact directly with device I/O elements such as files, databases, sensors and communication services.

Sensor Config is a file that contains default sensor configuration information for all active sensors such as sampling rates and change thresholds.

Rule Sets are files that contain rules that the context engine reasons about. There is a default rule set that is generic to all group-context aware mobile applications. Mission-specific rule sets can be created, added and swapped as needed.

App Data is the physical storage for the context model and app-specific data.

Sensor 1 to *Sensor N* correspond to the components that receive data from sensors. They all implement the same interface so that sensors can be easily added to the system.

Comms Mech 1 to *Comms Mech N* correspond to the communication mechanisms that are used to send to and receive data from other members of the group. They all implement the same interface so that communication mechanisms can be easily added to the system. Each communication mechanism has a configuration file that corresponds to communication-mechanism-specific information such as local addresses, server/router addresses, predefined user names-device pairings, ports, and security keys.

4 Architecture Decisions

There were several architecture decisions that were made to support the required capabilities listed in Section 2 and the extensibility scenarios presented in Section 3. Even though some of these decisions were made using Android-specific programming constructs and technologies, we argue that they can be implemented using equivalent technologies on other platforms.

4.1 Context Model "At the Center"

Given the need to support easy addition of sensors, communication mechanisms, events, and views, a decision was made to place the context model "at the center." This means that the context engine, views, sensors and communication are all based on producing and consuming context data defined by the context model as well as events that are generated based on changes in context data.

The goal established for the context model was to be generic and extensible in order to handle a wide range of situations, environments, and data.

Logical Data Model. The logical data model, or form and structure of the context model, is based on the definition of context provided in Section 2. This work expands the definition of an entity — originally stated as a person, place or object — to include three group-related entities: people (individuals, groups, and organizations), activities, and events. The high-level context model is shown in Fig. 2.

People. The novel contribution of this work is the expansion of the scope of context from the user to the group. As mentioned earlier, in tactical settings individuals rarely

work alone. In most cases, groups of varying size collaborate on tasks to achieve group-level goals (i.e., missions). Therefore, supporting this coordination requires effective sharing of context data between group members.

In the proposed group-context model, the Person entity is changed to People and divided into three subcategories: *Individual*, *Group* and *Organization*. This allows more fine-grained control over how to process each subcategory. Similarly, this breakdown allows further decomposition into different types of groups or organizations.

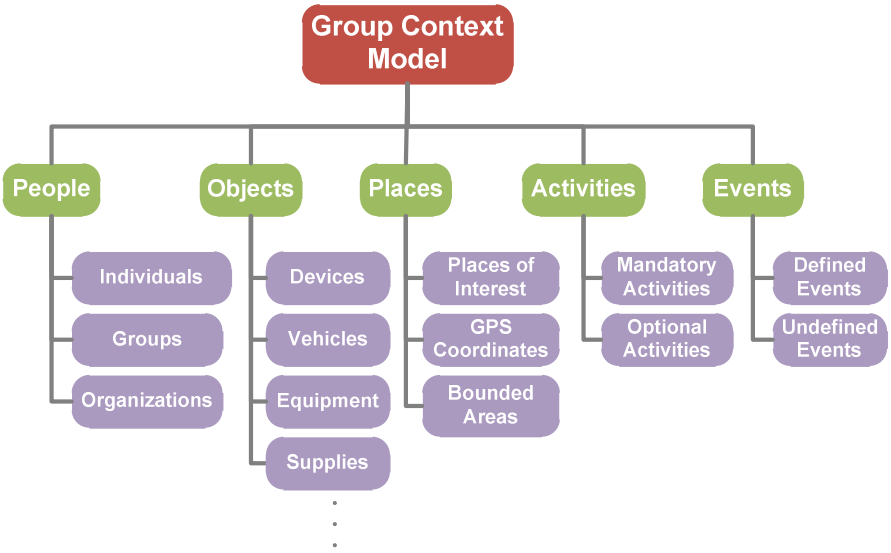

Fig. 2. Group context model

Activities. Activities are what individuals are doing, either on their own or as part of a group. Activities have status (complete, not started, in progress, on hold, etc.) that can be used to model task flow, react to changes in activity status, suggest or assign new activities, and provide information relevant to current activities.

Mandatory Activities are activities that are assigned to an individual and that must be completed (i.e., a task). *Optional Activities* are activities that an individual is performing that do not necessarily require status reporting. This differentiation can be used for tracking activities in a group that are part of a larger mission.

Events. Events are notifications related to changes in context data. Defined Events are set, known events that can either be detected programmatically because of changes in context data that generate the events, or input by users as an external event that can only detected by humans. The application has pre-defined responses for these events. For example, views can subscribe to these pre-defined events or the application may determine that a certain event has to be communicated to other members of the group.

Undefined Events are random events that can only be identified organically by users, and do not have pre-defined responses (e.g., a falling building). As such, they are considered to be informational only and users must respond to them manually.

Physical Data Model. Two options were considered for persisting context data: tables using a standard SQL-based approach, and objects using an OODB (object-oriented database) or ORM (object-relational mapping) approach. This is an important decision because it affects a number of quality attributes, including data model extensibility, performance, power consumption, and scalability.

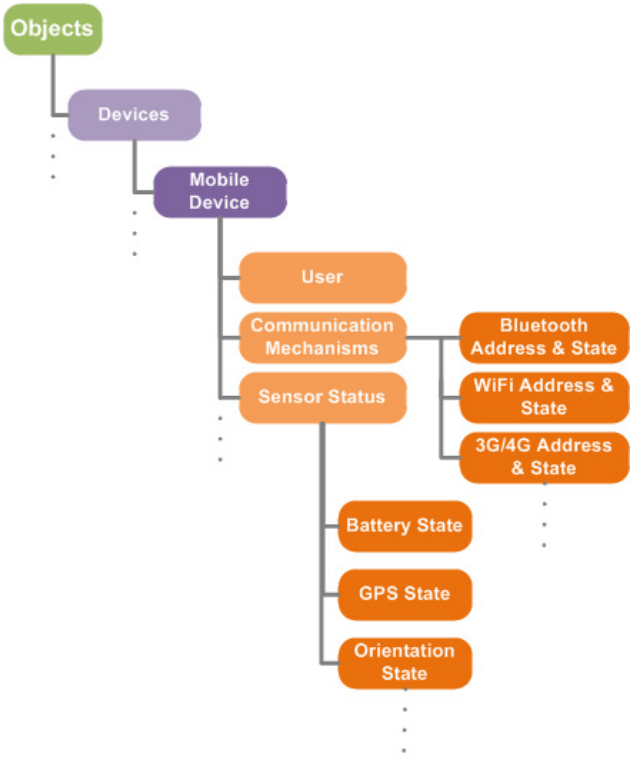

Fig. 3. Context model decomposition for *Devices*

Data model extensibility is better promoted by an OODB/ORM approach because tables do not need to be created or modified in order to support new types of context data. An OODB/ORM approach would either automatically create tables or eliminate them altogether, limiting the effort of extending the data model to that required to add the new context data element within the application. An example of an additional level of decomposition of the context model is shown in Fig. 3. Adding new devices would be a matter of adding a new subclass of Devices. All existing methods that operate on *Devices* would be applicable to the new class of device as well.

Performance and power consumption can be considered together because tests show that they scale together. In performance testing[1], results showed that an OODB implementation (Perst on Android [15]) performs orders of magnitude faster in most tests than an SQL-based implementation (SQLite on Android [16]), as shown in Fig. 4. As a result, less power is used by the OODB implementation, resulting in a correlation between increased performance and reduced battery consumption, as shown in Fig. 5.

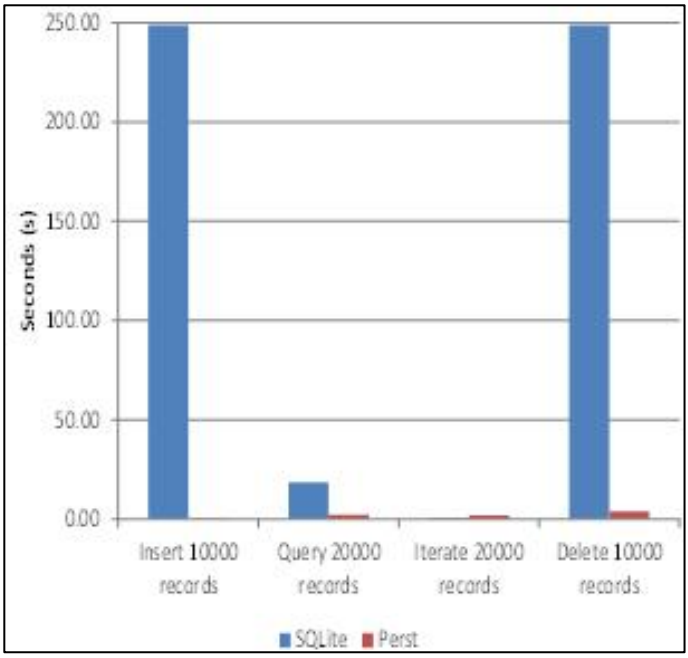

Fig. 4. Performance testing results for SQLite and Perst

The decision was therefore to implement the physical data model using an OODB because of the importance of supporting data model extensibility, given the expectation that the data model will change and evolve for the unique and varied situations first responders and warfighters might encounter at the tactical edge. Similarly, battery power is considered to be a key quality attribute at the tactical edge due to limited resources. The main tradeoff in this decision is scalability because in an OODB implementation most of the data is held in memory. Memory scalability is a concern because a large amount of data is captured by the mobile device sensors and shared between members of a group. The strategy for managing this concern is to limit data capture by disabling or throttling sensor usage as needed and limiting data sharing between group members to only relevant data and events.

[1] Tests were executed on the Android 2.3.4 platform with records of size 172 bytes.

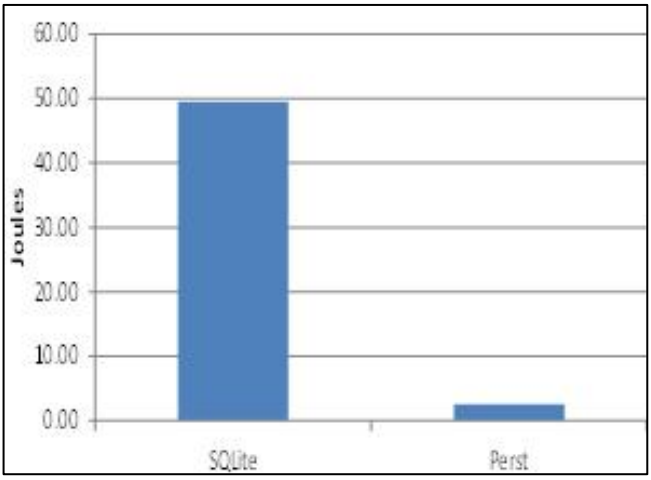

Fig. 5. Energy consumption for 1000-record benchmark

4.2 Context Sensors

To most effectively leverage context, it is important to capture as much of it as possible. As such, the architecture must not only support all available sensors currently onboard the mobile device, but also addition of new ones. To promote the extensibility scenarios related to sensors shown in Table 1, we adopted two architectural tactics: standardized interfaces and decoupling.

Mobile devices currently support a variety of sensors, from light and motion, to orientation, location, and proximity. As mobile devices grow more powerful and hardware components for new sensors become smaller and can be integrated into mobile devices, there will be a continuing need to integrate the inputs from new sensors into the application. To minimize the amount of effort to do so, we designed a standardized interface that any onboard sensor can use to report data to the application, as shown in Fig. 6. This limits the application effort to integrate a new sensor to the development of the sensor control logic (a single file), a few lines of code in the application itself (to add the new sensor and sensor value type, if previously undefined), and a new line in a configuration file (to set sensor defaults). We recognize, however, that as the number of sensors increases on mobile devices, so does the demand on device resources such as CPU, memory, and bandwidth if all sensors are capturing data. Therefore, we employed two approaches to limit the impact of sensor status reporting.

First, we implemented a sensor management capability to enable, disable, and throttle sensor status reporting as needed. This capability can be invoked from a user view or as needed by the application (e.g., based on context rules). Second, we

implemented a strict decoupling paradigm. All sensors are started as Android Remote Services (bound services) [17], which act as entirely separate background "applications" with their own memory space. This means that the processes gathering sensor information are decoupled from the main application while following Android's defined service life cycle [18]. Therefore, even if one or more sensor services are stalled due to high capture rates, the Android process management infrastructure will constrain the impact to the service and limit the resource impact on the primary application. In addition, it enables multiple consumers to use the same source of sensor information leading to an increase in flexibility in the layers above.

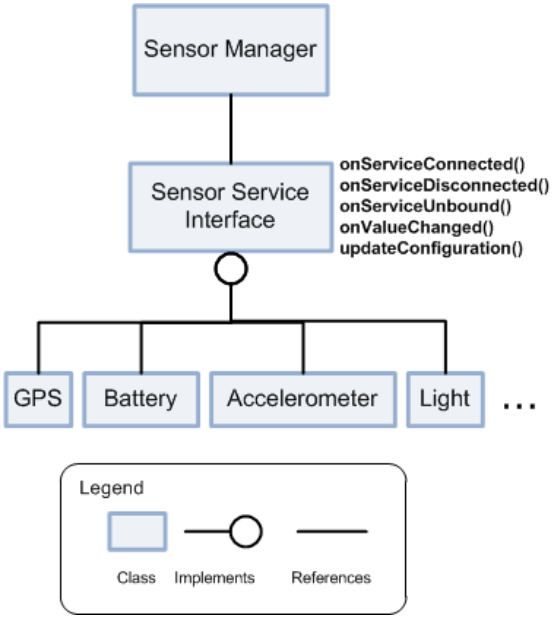

Fig. 6. Sensor service interface

The tradeoffs of this decision are added complexity, increased resource utilization, and performance overhead. Inter-process communication is done using an AIDL (Android Interface Definition Language) interface [19], which is an extra layer of infrastructure. This inherently increases the complexity of the system, making it harder to maintain. It also increases resource usage because the operating system is spawning an entirely separate process for each sensor. However, testing of the prototype with large workloads has shown performance and memory usage to be within acceptable parameters. With respect to maintainability, while this is a legitimate concern as the application evolves, we consider it to be a less important quality to promote than extensibility given the architecture drivers.

4.3 Context Dissemination

As important as it is to gather contextual data, group-context-aware mobile application requires robust context sharing capabilities. Because at the tactical edge it is difficult to predict what communication infrastructure will be available, the goal of the architecture is to support any available communication mechanism in a standardized way.

The architecture decision for enabling addition of new communication mechanisms is very similar to the architecture decision that supports adding new sensors: a standardized interface to easily integrate new communication mechanisms as they become available. The common service interface shown in Fig. 7 provides generic communication methods and callbacks that individual protocols can adapt as necessary. This approach enables the application to use any available communication channels as needed. If multiple channels are available, context information can be used to determine which channels are used and how.

The tradeoff of using a standardized interface for all communication mechanisms is that it implements the lowest common denominator of communication mechanism capabilities. For example, a connection-based protocol establishes communications and controls data transfer in a very orderly, systematic way. A connection-less protocol is much more ad-hoc and cannot be restricted to a specific sequence of events. Therefore, the interface has to support a very general connection process that can allow any sequence of communication events.

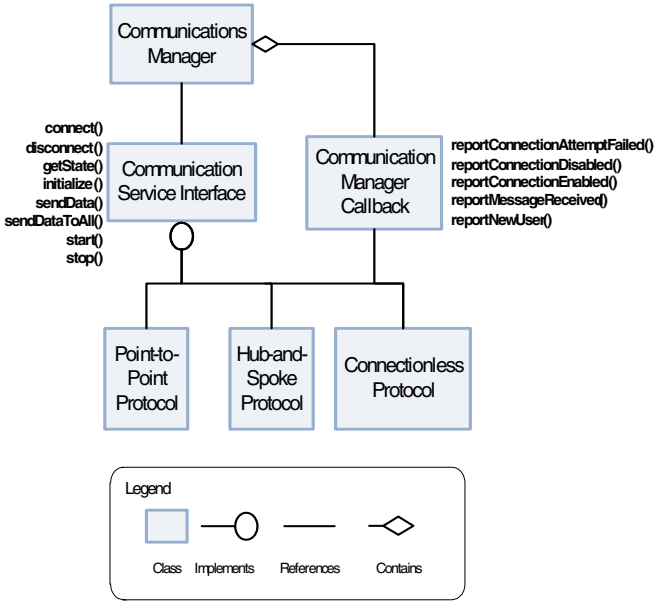

Fig. 7. Communication service interface

Another tradeoff is that in order to intelligently share context data, the application requires the ability to send data to specific users, regardless of the underlying protocol. For connectionless protocols, this could be difficult to support because the underlying protocol may have no knowledge of which user or device is the source of a particular message. It may also lose track of the message target status and actual data might get lost because message transmission is not guaranteed.

4.4 Context Engine

A challenge for any context-aware application is that for any given user, the relevance of any piece of contextual information entirely depends on the situation of the user. For example, the contextual information that a person cares about while at work is significantly different than the information they care about while they are at home, on vacation, playing sports, spending time with family, etc. Therefore, in order to construct a general-purpose group-context aware application, its architecture needs to enable the application to easily switch between different situations so that the relevance of a given piece of contextual data can be tailored to the user as well as the situation that the user is in.

A decision was made to encode all rules in external XML files (rule sets). The set of rules that the context engine reasons about can be changed by loading a different rule set, either at design time or runtime, the latter reflecting the reality of changing situations. A sample rule is shown in Fig. 8.

```
<Rule
 RuleName = "LocationUpdate"
 TriggerDataType = "GPSSensorDataUpdateEvent"
 Conditionals = ""
 Actions = "DECLARE PrimaryDeviceLocationUpdateEvent"
/>
```

Fig. 8. Sample context rule based on a location update event

Each rule in the rule set is evaluated only against a specific type of data item (or subscription object). The triggering data type is specified in the rule by the *TriggerDataType* field. Thus, when a data item is passed to the context engine for processing, only rules that are triggered by the data type of the data item are executed.

Evaluation of the rule is based on the evaluation of conditional statements included in the *Conditionals* field of the rule. The convention of a simple, semi-colon-delimited string is used so that rules can contain an arbitrary number of conditionals in any combination. Evaluations currently supported by the engine include the standard mathematical operators >, <, >=, <=, =, and /=. Conditionals support

comparison to a specific value, written in the rule as a number, or to a field of another object, such as the object that triggered the rule.

The last part of the rule is the *Action* field. There are different types of actions that an application could take if a rule is evaluated to be true based on the receipt of a new data item. For example, the application might need to send data to one or more users in the current group. Alerts might be displayed to users based on the criticality of the information. It is also possible to change the current active rule set so that the application can react to significantly different situations.

As an example, the rule in Figure 8 is called *LocationUpdate* and states that if a local location update is passed to the application by the onboard GPS sensor (this is indicated by the receipt of a data item of type *GPSSensorDataUpdateEvent*) then in ALL situations (no conditionals are evaluated) the application should declare a new event to indicate that the location of the device has changed. The action is that a *PrimaryDeviceLocationUpdateEvent* object is created, containing the new GPS coordinates, and this data is forwarded to all external connections (the encoding of the action is included at the end of the rule set file).

Another example of rule that can be expressed using the same XML structure is shown in Fig. 9. In this rule, called *BatteryLow*, the battery sensor reports that there has been a change in battery level by triggering a *BatterySensorDataUpdateEvent*. The rule looks at the *batteryCharge* field of this event to check if it is less than (*LT*) *30%*. Of this is the case, two actions are taken: one is to create an *ALERT 1* and the other is to declare a *BatteryLowEvent* (as with the previous example, the encoding of the action as well as the alert is included at the end of the rule set file). The full context rule notation and context rule engine implementation is the topic of an upcoming paper.

```
<Rule
 RuleName = "BatteryLow"
 TriggerDataType = "BatterySensorDataUpdateEvent"
 Conditionals = "TRIGGER.batteryCharge LT 30"
 Actions = "ALERT 1; DECLARE BatteryLowEvent"
/>
```

Fig. 9. Sample context rule based on a change in battery level

5 Prototype Implementation

To demonstrate the validity and extensibility of the reference architecture, we built a prototype application on the Android platform that implements a group context-aware mobile application for first responder task management. The concrete architecture is shown in Fig. 10.

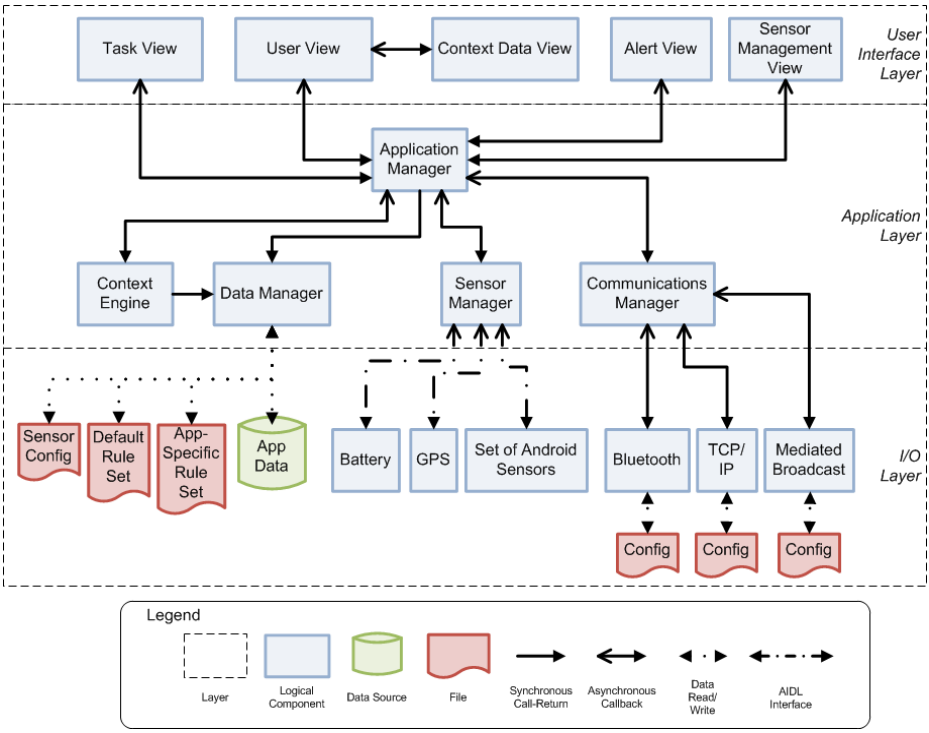

Fig. 10. Architecture for first responder task management prototype implemented on Android

5.1 Views

All views are implemented as Android activities. There are currently three views that we envision would be found in any group-context-aware mobile application:

- *User View*: Displays the list of users that are currently part of the group, as well as any relevant status or other contextual information. This view subscribes to events that indicate user list changes, such as connection events or group addition/modification events.
- *Context Data View*: Displays all relevant context data for a selected member of the group. It is invoked by the *User View* when a user is selected. Data for this view is retrieved from the *App Data* store.
- *Alert View*: Displays alerts to the user. This view subscribes to alert-reporting events.

The *Sensor Management View* was created as part of the prototype to enable manual adjustment of sensor sampling rates and understand the effects of changes in sampling rates. It also serves as an example of a view that changes context data that could also be changed programmatically by a rule (e.g. reduce sampling rates for all sensors when battery <=20%).

The *Task View* is an application-specific view. Groups of first responders are commonly task-based. That is, a leader of the group assigns a task to members of the

group and members report on the status of the tasks assigned to them. This view displays task-related information and responds to task-related events, such as new tasks or task status changes.

5.2 Sensors

Battery, *GPS*, and the *Set of Android Sensors* [20] were added by implementing the *Sensor Service Interface.* The context data elements that map to the information coming from these sensors were added to the context data model. Sensors report changes to their corresponding context model elements. The effort required to add new sensors proved to be as expected (see Section 4.2).

5.3 Communication Mechanisms

Bluetooth and *TCP/IP* communication was added by implementing the *Communication Service Interface*. To test the extensibility of the architecture and its applicability to connectionless protocols, we successfully integrated a *Mediated Broadcast* protocol implemented by research collaborators at George Mason University (GMU)[2]. Effort was within the parameters defined by the corresponding scenario.

5.4 Context Engine and Rules

Tasks were created as *Mandatory Activities* in the context model. *Mandatory Activities* have six fields/attributes: *name*, *participants*, *sub-activities*, *status*, *assigners*, and *assignees*. In addition to the *Default Rule Set* that applies to the three default views (*User*, *Context Data*, and *Alert*), a rule set was created for task management to react to task creation as well as changes in task status (*App-Specific Rule Set*). As new tasks were created these were assigned and sent to appropriate group members. As task status was updated, events were triggered to communicate status updates. An example of an app-specific rule for this application is to evaluate the status of any parent activities of the sub-activity that was completed: if all sub-activities are complete, then set the state of that parent activity should be set to complete as well.

6 Related Work

There is a large amount of work related to architecture and design of mobile context-aware applications. The most-referenced publication in this area is by Dey et al [2]. This work presents a *Context Toolkit* that supports common features required by

[2] The GMU Mediated Broadcast is a location-based ad hoc networking mechanism. It was selected to accommodate situations in which messages are targeted at a specific location rather than an individual (e.g., alert everyone to a disaster situation coming to the area). The mobile devices along the path serve as re-transmitters for the message. This work will be published in an upcoming joint report.

context-aware applications: capture and access of context, storage, distribution, and independent execution from applications. Our work extends the definition of context to include group-related entities as well as the capabilities of the main components of the toolkit (widgets, interpreters and aggregators) to process this group context. It also adds a context dissemination component to share context with group members. Baldauf et al [10] conducted a survey of multiple approaches for architecting context-aware systems and determined that it largely depends on how the system captures context data. It also claims that separation of detecting and using context is necessary to improve extensibility and reusability of a system. The proposed reference architecture is consistent with these two statements. Most other research in this area presents ideas and frameworks for designing context-aware systems, but is limited to individual context [3][7][9][12][24][25]. For example, Cagalaban and Kim [9] present an architecture of a context-aware system that includes web services, mobile devices, smart homes, and the different responsibilities in a health care context. The use of a rule engine is discussed for processing context in an individual system. As another example, Henricksen and Indulska [12] propose a notional layered architecture that uses separation of concerns, synchronous and asynchronous communications, and other architectural constructs to manage the capture and processing of contextual information in an individual system setting.

Commercial, location-based apps running on smartphones can filter information based on a user's current position, but do not use other information useful in tactical environments such as the state of individuals or their devices. Work in the commercial sector is just beginning to use richer context information from multiple individuals to inform handheld users and make recommendations (e.g., finding a possible restaurant for a group lunch) but do not treat the group as a collaborating resource. In addition, as with the related research, context is mainly limited to location and interests as part of social networks. Examples include SocialFusion [21], CenceMe [22] and Serendipity [23].

7 Conclusions and Future Work

Handheld mobile technology is reaching first responders and soldiers in the field to help with mission execution. Using this technology to sense as much of the environment as possible, share this information with members of a group, and reason about the group context can improve the effectiveness of the mission and optimize resources. In this paper we have described a highly-extensible reference architecture for group-context-aware mobile applications that integrates the contextual information from individuals with that of nearby team members.

Context is extended beyond the location and time of an individual to consider group context elements such as activities and events. A variety of sensors and communication mechanisms can be integrated into the architecture to account for the variability in devices as well as communication mechanisms that are available. A context engine based on mission-specific rule sets enables applications to reason about context and make decisions on what information to show to individuals, what

information to share with the group, and optimize individual and group resources. The next step is to implement a disaster recovery scenario and corresponding rule set in which resource-poor first responders are able to coordinate their efforts by sensing and sharing context information that is automatically filtered based on mission parameters. Once the scenario implementation is stable we will attend experimentation events where we will be able to test the architecture in a simulated setting and see how the architecture behaves under close-to-real infrastructure and data rates.

One of the more interesting results of this work has been the ability to leverage the architecture to support collaboration. By identifying extensibility scenarios early on in the design process, we were able to construct an architecture that supports multi-organizational collaboration to construct and evaluate different pieces of the architecture. This has allowed us to reach out to researchers from multiple universities and industry, resulting in synergistic research and development, furthering the goals of all participants.

Current and future work includes:

- Group context-aware resource optimization: use group context information to optimize battery consumption, bandwidth and computation resources of a group. Examples include the use of device status to assign computation-intensive tasks to devices with the most resources; adjustment of sensor input rates depending on battery level; and selection of appropriate communication mechanisms for context dissemination depending on comms status, information sensitivity and battery level.
- Minimizing user interaction for context capture: exploit device sensors to capture context information in a non-intrusive manner such that there is minimum disruption to the user's attention and activities. An example includes the use of the accelerometer.

Acknowledgements. 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-0000029).

References

1. Alabaster, J.: Japan's Softbank to offer world's first phone with radiation detection (2012), http://www.itworld.com/278997/japans-softbank-offer-worlds-first-phone-radiation-detection
2. Dey, A.: Understanding and using context. In: Personal and Ubiquitous Computing, vol. 5, pp. 4–7. Springer-Verlag London Ltd. (2001)
3. Godbole, A., Kim, S.-Y.: User centered design of context aware cell phones in human-centric systems. In: 2010 IEEE International Conference on Information Reuse and Integration (IRI), August 4-6, pp. 189–194 (2010)

4. Chen, H., Finin, T.: An ontology for context-aware pervasive computing environments. In: Proc. IJCAI Workshop on Ontologies and Distributed Systems, IJCAI (2003)
5. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, pp. 85–90 (December 1994)
6. Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Van Laerhoven, K., Van de Velde, W.: Advanced Interaction in Context. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 89–101. Springer, Heidelberg (1999)
7. Malek, J., Laroussi, M., Derycke, A., Ghezala, H.B.: Model-driven development of context-aware adaptive learning systems. In: Proc. 2010 10th IEEE International Conference on Advanced Learning Technologies, pp. 432–434 (2010)
8. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Dartmouth Computer Science Technical Report. TR2000-381 (2000)
9. Cagalaban, G., Kim, S.: Context-Aware Service Framework for Decision-Support Applications Using Ontology-Based Modeling. In: Kang, B.-H., Richards, D. (eds.) PKAW 2010. LNCS (LNAI), vol. 6232, pp. 103–110. Springer, Heidelberg (2010)
10. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2(4), 263–277 (2007)
11. Weerasinghe, T., Warren, I.: Odin: context-aware middleware for mobile services. In: Proc. 2010 6th World Congress on Services, pp. 661–666 (2010)
12. Henricksen, K., Indulska, J.: Developing context-aware pervasive computing applications: models and approach. Pervasive and Mobile Computing 2(1), 37–64 (2006) ISSN 1574-1192, doi:10.1016/j.pmcj.2005.07.003
13. Mehra, P.: Context-aware computing: beyond search and location-based services. IEEE Internet Computing, 12–16 (March/April 2012)
14. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison Wesley SEI Series in Software Engineering (2003)
15. McObject, Embedded Database for Android – Perst Small Footprint DBMS (2012), http://www.mcobject.com/android
16. SQLite.org, SQLite Home Page (2012), http://www.sqlite.org/
17. Android Developers, Bound Services | Android Developers (2012), http://developer.android.com/guide/topics/fundamentals/bound-services.html
18. Android Developers, Services | Android Developers, (2012), http://developer.android.com/guide/topics/fundamentals/services.html#Lifecycle
19. Android Developers, Android Interface Definition Language (AIDL) | Android Developers (2012), http://developer.android.com/guide/developing/tools/aidl.html
20. Android Developers, Sensor | Android Developers, (2012), http://developer.android.com/reference/android/hardware/Sensor.html
21. Beach, A., Gartrell, M., Xing, X., Han, R., Lv, Q., Mishra, S., Seada, K.: Fusing mobile, sensor, and social data to fully enable context-aware computing. In: Proceedings of the Eleventh Workshop on Mobile Computing Systems Applications (HotMobile), Annapolis, MD (February 2010)
22. Miluzzo, E., Lane, N., Fodor, K., Peterson, R.A., Lu, H., Musolesi, M., Eisenman, S.B., Zheng, X., Campbell, A.T.: Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In: Proc. of 6th ACM Conference on Embedded Networked Sensor Systems (SenSys 2008), Raleigh, NC, USA, November 5-7 (2008)

23. Eagle, N., Pentland, A.: Social serendipity: mobilizing social software. IEEE Pervasive Computing 4(2), 28–34 (2005)
24. Li, X., Lin, J., Li, L.: On the design of a mobile agent environment for context-aware m-commerce. In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), July 9-11, vol. 3, pp. 176–180 (2010)
25. Hsu, H.J., Wu, S.Y., Wang, F.J.: Methodology to developing context-aware pervasive applications. In: Proc. Fifth IEEE International Symposium on Service Oriented System Engineering, pp. 206–213 (2010)