# Supporting Generalized Context Interactions

Gregory Hackmann[1], Christine Julien[2], Jamie Payton[1], and Gruia-Catalin Roman[1]

[1] Department of Computer Science and Engineering
Washington University in St. Louis
{ghackmann, payton, roman}@wustl.edu
[2] Department of Electrical and Computer Engineering
The University of Texas at Austin
c.julien@mail.utexas.edu

**Abstract.** In context-aware computing, applications' behavior is driven by a continually-changing environment. Mobile computing poses unique challenges to context-sensitive applications and middleware, including the ability to run on resource-poor devices like PDAs and the necessity to limit assumptions about the network. Though middlewares exist to provide context-awareness to applications, they do not address the limitations inherent in dynamic mobile environments. This paper discusses a lightweight approach to context-sensitivity that takes into account these considerations. We explore the use of modularization to tailor service discovery policies for applications, as well as leveraging existing language constructs to simplify creation and aggregation of different context types. We also discuss an implementation of these concepts, along with three sample applications that can automatically propagate changes in context to clients running on devices from mobile phones to desktop computers.

## 1 Introduction

Traditionally, context-aware computing refers to an application's ability to adapt to its environment. Calendar or reminder programs [1] use time to display pertinent notifications to users. Tour guide applications [2, 3] display information based on the user's current physical location. Still other programs implicitly attach context information to data, e.g., to research notes taken in the field [4]. Each of these applications independently gathers context information from the required sensors and tailors the provision of context.

With the increasing popularity of communicating mobile devices, context-aware computing has moved from a target environment of an autonomous device to a sophisticated network of connected devices, all providing context information to each other. This enables powerful applications that allow complex interactions across a dynamic network of heterogeneous devices. Presenting context to software engineers, however, has received little attention. Building context-aware applications like those above has required each developer to independently construct mechanisms to monitor and collect context information.

In this paper, we introduce CONSUL, a middleware solution that simplifies access to context information. By providing abstractions for complex network

transactions, we allow novice programmers to build applications that utilize context information collected from a heterogeneous environment. We significantly simplify the development task by removing the need to handle the intricate network programming necessary to collect the information and instead present an accessible yet expressive and extensible interface for using context information.

In the next section, we outline the requirements of a context monitoring middleware for dynamic mobile environments. Section 3 examines existing solutions. Section 4 details the architecture and implementation of CONSUL, and Section 5 discusses three sample applications developed with the middleware. In Section 6, we address relevant issues, including discovery mechanisms, the separation of discovery and sensing, and higher level concerns associated with context-gathering. Conclusions appear in Section 7.

## 2 Problem Definition

Context items are pieces of data sensed about the environment, e.g., location, temperature, link latency, etc. The environment is open, meaning hosts contributing context information can join or leave the network at any time. We assume a heterogeneous and dynamic environment containing resource-constrained devices such as environmental sensors, cellphones, PDAs, and laptops.

Programming the collection and monitoring of dynamic contexts can be burdensome. A programmer must identify the desired source, contact the provider, collect the context items, and interpret them. Typically, the developer must use network programming mechanisms that require knowing the identity and location of the provider. In open and dynamic environments, it is often infeasible to rely on such a priori knowledge. Mobility compounds the problem since the movement of context providers requires management of network disconnections. In addition, given the wide array of devices available and the multitude of applications that run on them, the collected pieces of context are likely to be in diverse formats that require unification. Finally, the set of available context items is not static; applications continuously inject context items into the environment.

We aim to simplify application development by reducing the complexity of handling context collection and monitoring in dynamic environments. We achieve this goal through a middleware that hides the details of these tasks. The following are requirements of such a middleware infrastructure.

- **Decoupled communication.** We must assume no advance knowledge of communication partners.
- **Transparent monitoring of context.** Issues associated with distribution, mobility, and unpredictable connectivity should be hidden. Moreover, the process of determining how context changes are presented should be relegated to the infrastructure.
- **Generalized treatment of context.** Context should be generalized so applications interact with different types of information in a similar manner.

- **Extensibility.** Given the openness of the environment, the infrastructure should adapt to the inclusion of new context users and providers with little or no intervention from a system administrator.
- **Scalability.** To scale to large networks, a decentralized solution is necessary.
- **Accommodate small devices.** The middleware primitives must have a lightweight implementation to account for resource-constrained participants.

In the remainder of the paper, we examine how current solutions fall short of meeting these requirements and propose a new middleware infrastructure designed to facilitate rapid development of context-aware applications.

## 3 Related Work

In this section, we review examples of systems which support context-aware application development. We focus on three well-known systems: Stick-e Notes, CALAIS, and the Context Toolkit. For brevity, other context-aware systems such as CoolTown [5], Gaia [6], and Confab [7] are not discussed.

### 3.1 Stick-e Notes

Stick-e Notes [8, 9] favors ease-of-use and serves as a precursor for many context-sensitive middlewares. In Stick-e Notes, virtual notes are attached to physical phenomena like times, places, and events. The decision of when a note is in context is included within the note itself; the SGML structure of each note includes a section to semantically describe when it is to be triggered. Exactly what it means to trigger a note is left to the discretion of the client application.

This model is unique in that end-users need only basic SGML knowledge to create notes. However, significant trade-offs are made for the sake of ease-of-use. First, context is determined by the note and not the client, which limits flexibility. For example, a note may be triggered when the user enters a range of locations, but it is not possible for a user traveling in a car to trigger notes within a greater range of distances than a user on foot. In addition, the model provides no way to disseminate notes; the client either have them or be able to obtain them using some external mechanism. This limits the applicability of this model to dynamic environments.

### 3.2 CALAIS

CALAIS [10] offers an alternative for providing location-based context by allowing applications to register with sensors to receive notifications of state changes. Location is stored in a central database, which tracks physical objects (like Active Badges [11]) and uses spatial algorithms to determine which room contains these objects. This location information is automatically delivered to registrants. A simple language allows contexts to be aggregated into more-complex contexts.

The use of callbacks and context aggregation addresses the most serious shortcomings of Stick-e Notes by allowing clients to determine context from a number of sources, which automatically notify the client of any state change.

CALAIS relies extensively on CORBA, which too heavyweight for practical use on many mobile devices. Additionally, it is geared for a specific type of contextual information. Finally, the design of the location service necessitates a central server capable of processing complex spatial relationships, which raises additional performance and scalability issues.

### 3.3 Context Toolkit

The Context Toolkit [12] provides hooks for automatically discovering context-providing "widgets", which can be aggregated within the middleware to form more complex contexts. Unlike CALAIS, Context Toolkit does not depend on any specific back-end for communication between devices; by default it uses XML over HTTP for communication, but this can be swapped out to accommodate other communication mechanisms.

This model is not without its own shortcomings. First, the Context Toolkit is large and complex, which limits its use on resource constrained devices. This complexity also hinders the task of creating new widgets [13]. Finally, the movement of context aggregation functionality away from the client and into the middleware unnecessarily limits the types of aggregations that can be performed.

### 3.4 Observations

Despite their shortcomings, these systems identify several desirable characteristics of context-sensitive middleware. These characteristics, further refined in [14], form a list of challenges to meet when writing such a middleware. First, the context providing infrastructure must be independent of platform and programming language. Second, the system should adapt to changing context resources. Moreover, the system should adapt to changing context information and propagate these changes to applications. Third, the infrastructure must require minimal administration to be able to scale to large numbers of devices. Fourth, context should be treated universally to promote code reuse. Finally, to allow incorporation of resource-constrained devices, the middleware must remain lightweight.

## 4 A Middleware for Environmental Monitoring

Existing solutions fall short of meeting application needs, specifically on resource-constrained devices in highly dynamic networks. To address these concerns, we developed CONSUL (CONtext Sensing User Library), which provides application developers access to context information through a simplified interface. This eases programming and places the ability to build context-aware applications in the hands of novice programmers. Figure 1 shows CONSUL's architecture. In the

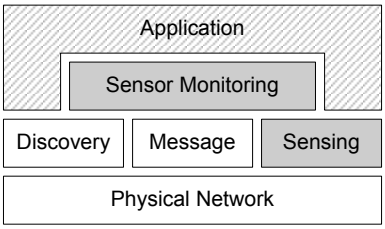


**Fig. 1.** The architecture of an application using CONSUL.

figure, the solid gray components define CONSUL. The white components we assume to exist, and the cross-hatched component is what an application developer provides. In this section, we discuss the implementation of these components and show how developers use CONSUL to build context-aware applications.

### 4.1 Foundational Components

In building CONSUL, we assumed the existence of several components. First, CONSUL builds on a physical network which includes the physical hosts and the connections (wired or wireless) that allow the hosts to communicate. On top of this, our middleware also relies on an existing message passing mechanism. The final component that we assume to exist is a network discovery mechanism. For the remainder of this paper, unless otherwise explicitly specified, we rely on the simplest discovery mechanism: one that informs a host of all one-hop neighbors, i.e., other hosts in direct communication. We intentionally separate discovery from sensing to allow each application to select its own discovery mechanism. We further examine this choice and its separation from context sensing in Section 6.1.

### 4.2 CONSUL

As shown in Figure 1, two components contribute to providing the environmental monitoring functionality: the sensing component and the sensor monitoring component. Figure 2 shows the internal class diagrams for these two components and how they interact with each other and the application.

**Sensing.** The sensing component allows software to interface with sensing devices connected to a host. Each device has a corresponding piece of software (a *monitor*). In CONSUL, each monitor extends an `AbstractMonitor` base class and contains its current value in a variable (e.g., the value of a location monitor might be represented by a variable of type `Location`). An application can react to changes in monitor values by implementing the `MonitorListener` interface and registering itself with the monitor. To ensure that any listeners registered for changes, the monitor should perform these changes through the `setValue()` method in the base class. Applications can also call the `getMonitorValue()` method provided by the base class to obtain these values on demand.

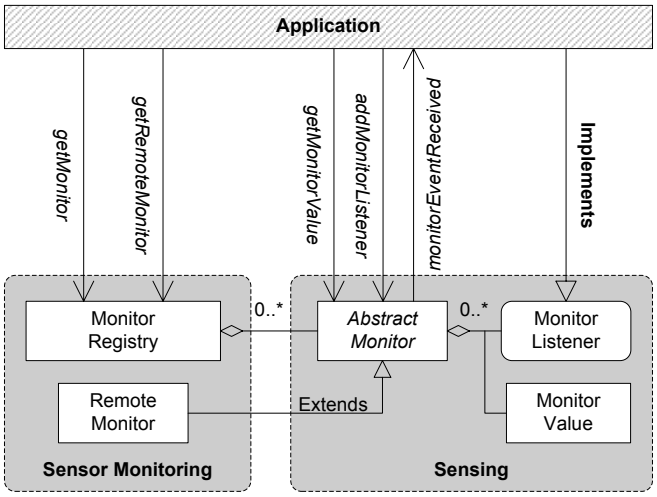


**Fig. 2.** The internal class diagrams for the components of CONSUL.

Figure 3 demonstrates an example class that extends `AbstractMonitor` to collect GPS information. From CONSUL's perspective, the important pieces are how the extending class interacts with the base class. The details of communicating with a particular GPS device are omitted; their complexity depends directly on the particular device and its programming interface.

```
public class GPSMonitor extends AbstractMonitor{
  public GPSMonitor(...){
    //call the AbstractMonitor constructor
    super("GPSLocation");
    //set up serial connection to GPS receiver
    ...
  }
  public void serialEvent(SerialPortEvent event){
    //handle periodic events from GPS receiver
    ...
    //turn GPS event into a GPSLocation object
    ...
    //set local value variable, notify listeners
    setValue(gpsLocation);
  }
}
```

**Fig. 3.** The GPSMonitor Class.

To assist application developers, CONSUL includes several `MonitorValue`s for programmers to use when building monitors or constructing more complex `MonitorValue`s. These values reside in a library to which application developers

```
public class GPSLocation extends ArrayValue {
  public GPSLocation(double latitude, double longitude) {
    super(new IMonitorValue [] {
     new DoubleValue(latitude), new DoubleValue(longitude)
    });
  });
  public double getLatitude() {
    return ((DoubleValue)getValues()[0]).getValue();
  }
  public double getLongitude() {
    return ((DoubleValue)getValues()[1]).getValue();
  }
}
```

**Fig. 4.** The GPSLocation Class.

can add new types. For example, the library contains an `IntValue` that can be used for sensors whose state can be represented as a single integer value. There are also aggregate values, e.g., `DateValue`, that build on the simple value types. In addition to being available for developers to use, they also serve as examples for defining new values. Figure 4 shows a class that extends `ArrayValue` to aggregate GPS coordinates (represented by `DoubleValue`s).

**Sensor Monitoring.** The sensor monitoring component maintains a registry of monitors available on the local hosts (*local monitors*) and on hosts found by the discovery package (*remote monitors*). As described above, local monitors make the services available on a host accessible to applications. To gain access to local monitors, the application requests them by name (e.g., "Location") from the registry, which returns a handle to the local monitor.

To monitor context information on remote hosts, the monitor registry creates `RemoteMonitor`s that connect to concrete monitors on remote hosts. These `RemoteMonitor`s serve as proxies to the actual monitors; when the values change on the monitor on the remote host, the `RemoteMonitor`'s value is also updated. To access remote monitors, the application provides the ID of the host (which can be retrieved from the discovery package) and the name of the monitor to the registry's `getRemoteMonitor()` method. This method creates a proxy, connects it to the remote monitor, and returns a handle. The application can then interact with this handle as if it were a local monitor.

## 5 Example Applications

In this section, we present three applications, and for each show how using CONSUL extensively simplified the programming task.

### 5.1 Stock Viewer

In the first application, stock quotes are delivered to handheld devices. Behind the scenes, servers advertise stock information by acting as monitors. Clients

running on J2ME-enabled devices automatically discover advertised stocks and display them to users who can select a stock and view its current value. Implementing the stock ticker using CONSUL is straightforward and requires minimal "from scratch" coding.

Since we do not have access to a J2ME-enabled mobile phone, the screenshots below are taken from a laptop running the MIDP Emulator from Sun's J2ME Wireless Toolkit. The emulator uses an 802.11b wireless connection to communicate with a stock server on a desktop computer. To simulate a low-bandwidth connection, the emulator caps the network throughput at 9600 bits/second.


**Fig. 5.** Left: the client stock ticker on a mobile phone emulator, displaying a list of discovered stock monitors. Right: the client displaying the value of a selected stock.

A stock's value consists of its ticker symbol; current, low, and high dollar values; trading volume; and company name. Creating a custom `StockValue` class simply requires aggregating the predefined `StringValue`, `IntValue`, and `DoubleValue` classes in an `ArrayValue`.

The stock monitor inherits its ability to automatically notify clients of changes from the `AbstractMonitor` base class. The implementation of `StockMonitor` requires only a call to `AbstractMonitor`'s `setValue()` method to update its value and propagate the update to clients. As shown in Figure 6, the server benefits from similar substantial code re-use. The simple code snippet shown assembles a fully-functioning context server from the CONSUL components and the two classes mentioned above. The first three lines start a device-discovery server. Then, a registry is created on a particular port ($p$) to allow remote hosts to query local monitors on port $p$. The final lines create local monitors for the MSFT, YHOO, and T stock tickers.

The client gains most of its functionality from CONSUL's discovery server and monitor registry components. Once the user interface code has been written,

```
DiscoveryServer discovery = DiscoveryServer.getServer();
discovery.setProxy(true);
discovery.start();
MonitorRegistry registry = new MonitorRegistry(p);

registry.addMonitor(new StockMonitor("MSFT"));
registry.addMonitor(new StockMonitor("YHOO"));
registry.addMonitor(new StockMonitor("T"));
```

**Fig. 6.** A Stock Ticker Server.

adding the stock querying functionality is almost trivial: four lines of code to begin finding stock servers, six lines to listen to monitors on discovered servers, and two lines to receive updated stock values. Such extensive code re-use allows rapid development of context-aware applications, shifting effort away from the back-end and toward the user interface.

In this application, the use of CONSUL decouples servers from specific clients, allowing for custom clients that can present stock information in various ways, e.g., a client that pops up a notification when a stock reaches a certain price.

Since CONSUL is very lightweight, the client can easily be run on resource-poor devices like mobile phones. The stock viewer application plus all required libraries consumes only 48 kilobytes. This small size also means that it is feasible to send the application to devices on-demand over-the-air.

The use of context-aware middleware places one restriction on the devices used as clients: they need full IP networking support. In this sample application, the server obtains stock information from a Web service. Web services have one distinct advantage over full-scale context-aware middleware: the clients only need basic HTTP networking support. This drawback does not necessarily reflect a general shortcoming of our middleware, but rather raises the issue of how useful context-sensitivity is in this scenario; stock information is widely available from well-known sources on the Internet. Despite our middleware's small footprint, from a practical point-of-view it may be overkill for retrieving stock quotes. The need for context-sensitive middleware is more pressing when the context sources are not public or must be discovered at runtime.

### 5.2 RFID-Activated Smart Room

In our second sample application, a server uses an attached RFID scanner to provide information about current occupants of a room. Information about a user's presence is collected by using wearable RFID tags and an RFID scanner on the door. A separate client stores predefined playlists for each person who might enter the room. This computer uses the continuously-updated list of people in the room to play music in the Winamp media player; it selects music from a "master" playlist made by intersecting the playlists of everyone in the room.

The client in turn serves as a monitor that provides information about the song currently playing, which can be displayed on a handheld. Another computer combines the MP3 player's context with the RFID context information to serve a Web page with a list of current occupants and the music.

The screenshots below show the application running on a Compaq iPaq and a desktopk, which communicate using 802.11b and wired Ethernet (respectively). Additional desktops (not shown) run the RFID and playlist server applications.

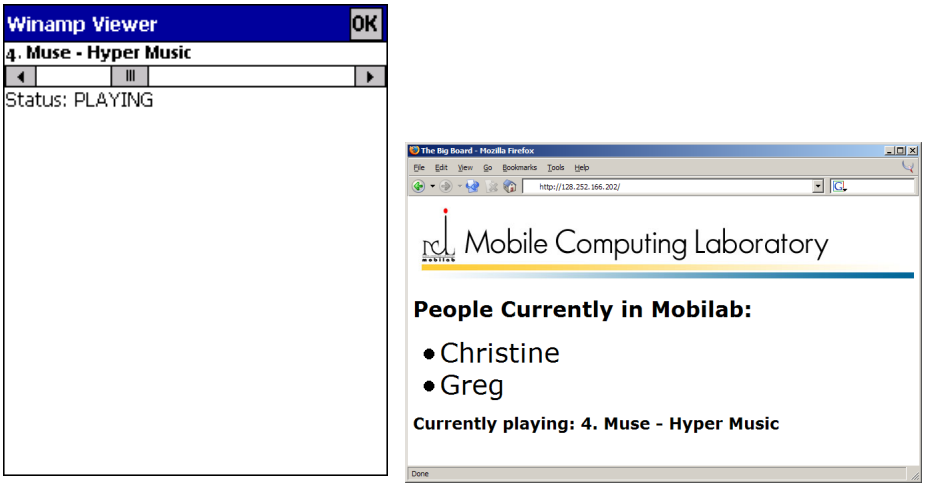


**Fig. 7.** Left: a client running on a PocketPC showing the current song being played. Right: a Web page showing the current song being played and the people in the room.

Unlike the previous example, these contexts are not publicly available on the Internet nor from well-known sources, so device discovery is essential. We also expect the playlist to be updated in real time as the room's occupants change, i.e., the client requires "push" service, which CONSUL provides; Web services inherently provide "pull" information.

The code to add context-sensitivity to this application is very similar to the previous application. Interestingly, the computer running the MP3 player acts both as a recipient of context information (a list of occupants) and as a provider (the current song). Once the RFID client was written, extending it to serve the MP3 player's contextual information required only a single line of code to instantiate a `WinampMonitor` and add it to the `MonitorRegistry` already in use to receive context from the RFID server.

Since these context monitors are re-usable components, extending this application is straightforward and transparent to the application. For example, a computer connected to an X10 controller could use the RFID context to automatically turn the lights off when the room is empty.

This application demonstrates that the monitors and values constructed using CONSUL are re-usable components just as CONSUL itself is. Multiple clients use the context provided by the RFID monitor and MP3 player monitor for different purposes, and the Web server drew context from multiple services. The `MonitorRegistry` class handles this transparently, so the programmer simply assembles applications from re-usable components.

Currently, the device discovery mechanism does not allow programmers to search for specific types of devices. Instead, clients retrieve a list of all the devices in the room. They must then collect a list of monitors running on the devices and select one or more monitors based on their names. This requires us to assume that a monitor's name reflects its function. For example, the clients interested in RFID information search for monitors named "RFID". This problem can be avoided by replacing the existing device discovery mechanism with a more-sophisticated discovery method as discussed in Section 6.1.

### 5.3 Ad Hoc Mobile Communication Protocol

The Network Abstractions protocol [15] provides context-sensitive routing in ad hoc networks by allowing an application to limit its operating context to a neighborhood within the ad hoc network. The protocol requires monitoring the values of sensors on the local host and on directly connected hosts in an ad hoc wireless network. To allow the size and scope of the neighborhood to be application-specific, each application can specify an abstract metric over arbitrary properties of hosts and links in the network. This metric calculates a logical distance from the application's local host to any other host, and includes a bound on allowable distances that restricts the hosts belonging to the neighborhood. As a simple example, an application might want to communicate with all hosts within three miles.

An implementation of this protocol can benefit from the use of CONSUL to access the properties of hosts and links that define metrics. CONSUL relieves the protocol implementer and user from concerns associated with maintaining a consistent view of the values of the relevant sensors on the local host and remote hosts. For example, when the protocol wants to build a new context that it maintains over time, it can use the code in Figure 8 to register itself as a listener for the appropriate monitors. When changes in the monitor values occur, the protocol is automatically notified and can change the structure of the routing paths as needed.

The code in the figure explains how, on behalf of a single application, the protocol uses CONSUL to build a network abstraction based on relative physical locations. The first line of the displayed code creates an instance of a monitor listener (the `ContextMonitorListener`). The protocol then retrieves the local instance of the `GPSMonitor` and adds the created listener to the monitor. This allows the protocol to be notified when the local host's location changes. Because this example metric is based on the physical distance between hosts in the network, the protocol must also register as a listener for changes in all the one-hop neighbors' locations. In the second portion of code, for each neighbor (in a

```
ContextMonitorListener cml = new ContextMonitorListener(...);
AbstractMonitor m = registry.getMonitor("GPSLocation");
m.addMonitorListener(cml);

for(int i=0; i<neighbors.length; i++){
   AbstractMonitor m2
            = registry.getRemoteMonitor("GPSLocation", neighbors[i]);
   m2.addMonitorListener(cml);
}
```

**Fig. 8.** A Portion of the Network Abstractions Protocol using CONSUL.

list retrieved from the neighbor discovery component), the application adds its listener to the remote location monitor on the neighbor. Not shown in this code snippet is the fact that, when new neighbors are discovered, a listener must be added to their location monitors, and when neighbors move away, the listeners must be removed. Additional code within the listener also handles the reception of monitor events to adjust the metric values when the locations of the involved hosts change.

### 5.4 Comparisons and Lessons Learned

These sample applications demonstrate CONSUL's flexibility. They include components running on desktops, PocketPCs, an emulated mobile phone, and a variety of other mobile devices. This flexibility comes from CONSUL's small footprint and the fact that it does not rely on any language-dependent features. In comparison, CALAIS and Context Toolkit have large footprints and would very likely not run on smaller devices like PocketPCs or mobile phones.

The applications presented in this section required little programming effort to transform a stand-alone utility or viewer into a context-aware application because CONSUL encapsulates the functionality needed to find and propagate context information across a network. To implement the same applications in Stick-e Notes would require additional code to propagate context to clients.

The applications demonstrate that CONSUL promotes separation of concerns, modularity, and code reuse. In the smart room application, custom-made monitors and values were effectively separated into re-usable components. A relatively complex smart room was built using simple components. This application also demonstrates that CONSUL promotes the development of extensible context-aware systems.

## 6 Discussion

In this section, we examine issues that arise in the use of CONSUL. The first concern deals with the underlying mechanism of network discovery. We then discuss the importance of the separation of concerns, focusing on the separation of

the two components within CONSUL and on the separation of CONSUL from network discovery. Another concern mobile computing developers express is the need to secure their information and devices. We discuss this in the context of sensor information that components make available. Finally, the CONSUL package provides a basis for building more sophisticated data interaction mechanisms, and we examine possibilities for these higher-level concepts.

### 6.1 Network Discovery

Well-known ad hoc mobile routing protocols generally use the simple network discovery mechanism assumed in Section 4. In these cases, all members of the network listen for any of their one-hop neighbors. However, this may cause problems in some target environments.

For example, conserving energy while discovering useful neighbor sets might be the driving design motivation. Birthday protocols [16] have been developed for static ad hoc networks where certain assumptions hold about the relationships between the devices. These networks are still quite dynamic, however, because nodes can be deployed and fail at various times, and require constant discovery.

Group communication mechanisms for mobile networks [17] can extend a node's neighborhood to include nodes to which it is not directly connected. Such protocols create a list of nodes with which a group member can reliably communicate. The integration of these group communication protocols with the CONSUL package allows applications to access sensor services available throughout the group instead of restricting remote sensing to one-hop neighbors.

Even in this brief overview, it becomes obvious that sophisticated discovery mechanisms can greatly enhance CONSUL. It is also apparent that the selection of the discovery mechanism depends heavily on a particular application's needs or operating environment. This factor plays heavily to our desire to separate the discovery mechanism from the CONSUL implementation.

### 6.2 Separation of Concerns

A key to software engineering is the identification and encapsulation of pieces of software related to a particular purpose. Software designed in accordance with the separation of concerns concept is highly modular and promotes code maintainability, reuse, and evolution. In CONSUL, we seek to provide a flexible and general middleware for context-aware application development in dynamic environments. As a result, CONSUL's architecture is highly modular.

Discovering the set of neighboring hosts that can contribute to an application's context is an important part of collecting context. In CONSUL, we separate the network discovery mechanism from the context interaction mechanisms (i.e., sensing and monitoring components). The discovery component constantly evaluates a host's set of neighbors. This neighbor set is used by other components in CONSUL to support context interaction. With this separation, we allow different methods of discovery to be used interchangeably without affecting how an application interacts with sensors.

In CONSUL, context interaction is further synthesized into components. The tasks associated with acquiring and reporting data are separated from those associated with acquiring sensors. The sensing component encapsulates sensing tasks (i.e., one-time and persistent query handling) within a monitor. The sensor monitoring component provides the application with a handle to monitors, local or remote, through the use of a monitor registry. With this separation, we place the responsibility on the sensor monitoring component for providing access to the desired monitors. Moreover, we eliminate the need for an application to use separate interfaces for interacting with local and remote monitors.

### 6.3 Security Concerns

CONSUL uses no encryption when sending monitor values between hosts. This avoids burdening resource-constrained devices with storing a large encryption library and decrypting values on-the-fly. However, monitors could be used to transmit sensitive data on insecure networks, justifying this burden. For these applications, an optional `EncryptedValue` class is included in CONSUL. This class wraps values with an encryption layer provided by the Bouncy Castle Crypto library [18]. Monitor programmers can use this class to encrypt values with a fixed symmetric key; access to monitor values can be controlled by distributing this key ahead-of-time to trustworthy clients. Clients without this key can still receive monitor values, but cannot decrypt them.

Since CONSUL's values are reusable components, the use of this generic wrapper adds only one line of code each to the monitor and client. Implementing more-sophisticated encryption schemes should be just as straightforward.

Unfortunately, this encryption layer does not address situations where untrustworthy clients should not know that certain monitors exist. This problem can be solved by incorporating access control into device discovery mechanisms.

### 6.4 Supporting Sophisticated Data Interaction

CONSUL provides simple data access; a request is issued for a monitor value, and the value is returned. Though CONSUL is lightweight, it can support high-level data handling and can be used to address data access issues important in context-aware application development.

Researchers are increasingly concerned with performing in-network data aggregation to reduce the number of responses funnelled to a query's issuer. Approaches include directed diffusion [19], Tiny Aggregation (TAG) [20], and digest diffusion [21]. CONSUL can support data aggregation by employing a hierarchicy of monitors. An aggregation monitor is at the top of the hierarchy, and is constructed by registering persistent queries on neighboring monitors or aggregation monitors of the same name. An application receives responses to queries only from the top-level aggregation monitor.

Another data access issue of mounting concern is imprecise data. Sensors will eventually fail, often resulting in incorrect or imprecise data reports. Recent

work has allowed applications to detect and respond to such imprecision. For instance, spatio-temporal relationships between sensor readings can be exploited in an online learning algorithm to predict current readings and detect abnormalities [22]. Other approaches quantify the tolerable level of uncertainty of a query and the level of uncertainty associated with a response [23]. A simple variation of these approaches could be employed using CONSUL, again through the use of a hierarchy of monitors. At the top of the hierarchy, a monitor registers a persistent query on another monitor of the same type and begins to calculate statistics about the monitor's readings. If the current reading is not within a specified threshold of the statistical expectation, the top-level monitor makes a decision about what to report to the application.

## 7 Conclusions

In this paper, we presented CONSUL, a lightweight middleware designed to support context-aware application development. CONSUL simplifies context interactions by encapsulating sensing and monitoring tasks, and provides a simple API for accessing context. In CONSUL, context monitors perform sensing tasks, and handle one-time and persistent queries issued by applications. An application uses a monitor registry to obtain monitors. Thus, the responsibility for obtaining the desired monitor from a constantly changing set of monitors is placed on the middleware, and the application can interact with sensors using a unified API.

The applications presented in this paper highlight how CONSUL meets requirements for middleware designed for dynamic mobile environments; it is portable, scalable, adaptable, and applicable to small devices. The applications presented demonstrate CONSUL's use on platforms ranging from desktop computers to mobile phones. Since CONSUL's underlying communication mechanism is very simple (message passing over standard TCP/IP sockets), if the device discovery mechanism used for an application scales well, then CONSUL scales no worse than a standard client/server application. Adaptability is demonstrated in the smart room application by CONSUL's use of device discovery to find and replace sources of context. Finally, CONSUL has an extremely small footprint, and can be used in applications on an emulated mobile phone, whose resources and feature set are extremely limited.

## References


1. Dey, A.K., Abowd, G.D.: Cybreminder: A context-aware system for supporting reminders. In: Proc. of the $2^{nd}$ Int'l Symp. on Handheld and Ubiquitous Computing. (2000) 172–186
2. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks **3** (1997) 421–433
3. Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.: Experiences of developing and deploying a context-aware tourist guide: The GUIDE project. In: Proc. of MobiCom. (2000) 20–31

4. Pascoe, J.: Adding generic contextual capabilities to wearable computers. In: Proc. of the $2^{nd}$nd Int'l Symp. on Wearable Computers. (1998) 92–99
5. Kindberg, T., Barton, J.: A Web-based nomadic computing system. Computer Networks **35** (2001) 443–456
6. Romn, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: A middleware infrastructure for active spaces. IEEE Pervasive Computing **1** (2002) 74–83
7. Hong, J.I., Landay, J.A.: An architecture for privacy-sensitive ubiquitous computing. In: Proc. of MobiSys. (2004) 177–189
8. Brown, P.J.: The stick-e document: a framework for creating context-aware applications. In: Proc. of EP'96. (1996) 259–272
9. Brown, P.J., Bovey, J.D., Chen, X.: Context-aware applications: from the laboratory to the marketplace. IEEE Personal Communications **4** (1997) 58–64
10. Nelson, G.J.: Context-Aware and Location Systems. PhD thesis, University of Cambridge (1998)
11. Want, R., Hopper, A., Falco, V., Gibbons, J.: The Active Badge location system. ACM Transactions on Information Systems **10** (1992) 91–102
12. Dey, A.K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, Georgia Institute of Technology (2000)
13. Yellin, D.M.: Stuck in the middle: Challenges and trends in optimizing middleware. SIGPLAN **36** (2001) 175–180
14. Hong, J., Landay, J.: An infrastructure approach to context-aware computing. Human-Computer Interaction **16** (2001)
15. Roman, G.C., Julien, C., Huang, Q.: Network abstractions for context-aware mobile computing. In: Proc. of the $24^{th}$ Int'l. Conf. on Software Engineering. (2002) 363–373
16. McGlynn, M.J., Borbash, S.A.: Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In: Proceedings of MobiHoc. (2001) 137–145
17. Huang, Q., Julien, C., Roman, G.C.: Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. IEEE Transactions on Mobile Computing **3** (2004) 192–205
18. Legion of the Bouncy Castle, The: The legion of the bouncy castle. http://www.bouncycastle.org/ (2004)
19. Intanagonwiwat, C., Govindan, R., D.Estrin, Heidemann, J., Silva, F.: Directed diffusion for wireless sensor networking. ACM/IEEE Transactions on Networking **11** (2002) 2–16
20. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: Tag: A tiny aggregation service for ad-hoc sensor networks. In: ACM Symp. on Operating System Design and Implementation. (2002)
21. Zhao, J., govindan, R., Estrin, D.: Computing aggregates for monitoring wireless sensor networks. In: $1^{st}$ Int'l. Workshop on Sensor Network Protocols and Applications. (2003)
22. Elnahrawy, E., Nath, B.: Cleaning and querying noisy sensors. In: Proc. of the $2^{nd}$ Int'l. Conf. on Wireless Sensor Networks and Applications. (2003) 78–87
23. Cheng, R., Prabhakar, S.: Managing uncertainty in sensor databases. SIGMOD Record, Special Section on Sensor Network Technology and Sensor Data Management **32** (2003)