sdlib: A Sensor Network Data & Communications Library for Rapid & Robust Application Development

David Chu
EECS Department
UC Berkeley
Berkeley, CA
davidchu@cs.berkeley.edu

Wei Hong
Arched Rock
Berkeley, CA
wei.hong@archedrock.com

***Abstract*—Sensor network applications tend to exhibit significant high-level commonalities along several major dimensions that have heretofore been underexposed. We are developing a component library, *sdlib*, which presents the fundamental abstractions, while operating efficiently.**

I. INTRODUCTION

Many useful abstractions have existed for some time in TinyOS, the sensor network operating system: GenericComm for link layer transmissions, Timer for timing events, and ADC for analog to digital conversions and sensor readings. However, these abstractions are low-level; the application developer still undertakes a sizable challenge when attempting to design her particular application. For example, suppose our application developer desires to build a best-effort video monitoring application. If the de facto standard nesC is the programming language chosen, at a minimum, the novice must master split-phase asynchronous programming, sidestep insidious race conditions, and gracefully handle resource contention. Moreover, the non-expert and expert developer both face significant challenges building plumbing for handling queries and network-wide data delivery.

It is unfortunate then, that our developer can only benefit minimally from another recently completed reliable-delivery vibration event detection application, which possesses both significant similarities (e.g. large data objects; query handling; Flash storage buffering) and differences (e.g. need for retries; monitored polling vs. event triggered) with her own. Yet in order to successfully make use of it, she must first know about the existence of this foreign application, entrust in its maturity, extract the relevant similarities and adapt them to suit her needs. Clearly this approach to reuse is not scalable (with the number of "reusable" applications), is error-prone, and is tedious.

Daunted by these obstacles, or simply by lack of knowing about pertinent similar applications, our application developer may choose to use high-level languages, such as TinyDB's TinySQL [1], SNACK [2], or Mate's various languages [3]. None of the aforementioned high-level languages provides direct support for large data objects, a fundamental requirement for our example application. In general, a high-level language is a suitable choice only if the user's task is within the scope of the chosen high-level language

We seek the middle ground. The goal of this work is to identify common functionality among a broad range of sensor network applications yearning for appropriate abstractions, and develop a library of thoroughly-tested, reusable and efficient nesC components that present the fundamental high-level operations while parameterizing essential differences. We call this library sdlib: Sensor Data Library. We draw an analogy to the traditional C++ STL. Sdlib provides powerful components for the reoccurring common cases. Simultaneously, because sdlib is implemented as a collection of nesC components, the developer retains unfettered access to low-level operations when desired.

Sdlib will not eliminate asynchronous operations, race conditions, or resource contention. Eliminating these usually incurs an unacceptable system penalty. Rather, sdlib enables the developer to relieve herself of a system full of such concerns and instead directs focus to the core application-specific module which can be more easily debugged. A set of composable components can greatly simplify the development task and mitigate the developer's worries.

Yet successful libraries offer generality without sacrificing efficiency. Efficiency of operations is particularly critical for sensor networks due to battery life, RAM/ROM and other resource constraints. Here sdlib exposes policy decisions such as resource allocation and rate of operation to the developer, while hiding the mechanisms of policy enforcement.

II. UNCOVERING PATTERNS

Our first task was to identify the various commonalities among existing sensor network applications. To gain an appreciation for the diversity of applications, we analyzed 13 sensor network applications: Nucleus [4], TinyDB, Deluge, Drip, Drain, Beacon Vector Routing (BVR), Tour Routing (TR), Directed Acyclic Graph Routing (DAGR), Base2Point Routing (B2PR), Synopsis Diffusion (SD), Golden Gate Bridge App (GGB), and Fabrication Equipment App (FAB), IMote2 Video App (VID) [5]. A fraction of these are not applications proper, but rather are service layers. Routing services have been abundant: BVR, Drain, DAGR, B2PR, SD. Others are dissemination services: Deluge, Drip. Of the applications, Nucleus and TinyDB provide general query engines to attributes. However, not all applications' needs are met by these existing systems and, as a result, application specific query engines arose: GGB, VID and FAB.

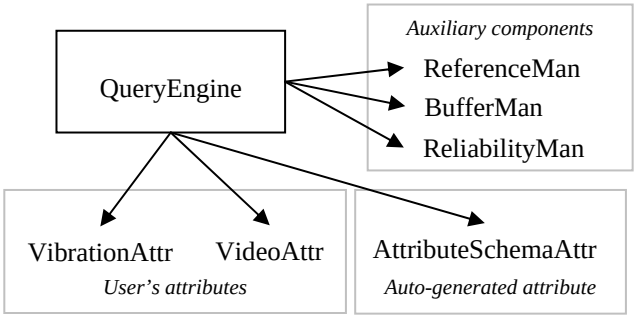

Figure 1. Component diagram of an example application using sdlib.

This is but one way to categorize these applications. After analysis, we extracted the following dimensions of variability:

- **Source & Destination**: Most of the applications offer only sending messages from the base station of a network along a spanning tree down to all nodes, or from one node up to the root. On the other hand, Deluge and Drip deliver messages to all nodes in the network. TR specifies a sequence of destinations. BVR offers virtual coordinate geographic coordinates.
- **Routing**: SD, TinyDB, Drain, DAGR and B2PR route along a gradient wrt the base station, whereas Deluge and Drip resemble intelligent global flooding. TR specifies a source route. BVR routes greedy-geographically.
- **Data size**: Deluge, GGB, FAB and VID must handle transport of large data objects whereas Drip, Nucleus and TinyDB handle only small objects.
- **Reliability**: GGB users demand reliable data transmission. On the other hand, Nucleus, TinyDB provide best-effort service.
- **Base station**: Several applications accord special status to the base station, though do so in different ways. Nodes in GGB, FAB, VID and TinyDB generate data for consumption by the base station whereas Deluge and Drip invert this relationship and cast the base station as the data generator with the nodes as data consumers.

The overlaps in these communication patterns appeared substantial, though the differences were also significant. We next chose several of these dimensions to initially incorporate in sdlib.

III. System Design and Implementation

We have completed an iteration of sdlib that initially supports (1) transporting large and small data objects and (2) unreliable and reliable data transport. Figure 1. shows the component diagram of an example application using sdlib. The query engine is the central point of control. The user's parameterization of the system optionally brings in auxiliary components, such as the `ReliabilityMan`, which handles acks and retransmissions. In this way, sdlib presents its library of services. The query engine receives queries through various communication channels and dispatches accordingly to the relevant attributes. The developer implements the appropriate attributes (often sensors) for her application. We built upon Nucleus' query engine, which provides support for unreliably transporting small data objects. Sdlib also optionally generates schema attributes that support run-time attribute discovery.

Sdlib exposes a simple, flexible, and efficient interface to the developer. Our developer, responsible for delivering a system that acquires results from a new vibration sensor (considered a large object), only needs to implement the logic of a simple interface for the `VibrationAttr` component. The two core functions are:

```
command result_t get(Buffer_t *pInitialBuf);

event Buffer_t*
givePiece(Buffer_t *pBuf, int len, bool last);
```

These require no more than the developer filling the buffer `pInitiialBuf` with sensor output when `get()` is called, and signaling `givePiece()` in response. A return value immediately provides a new buffer to which to copy more output, and issue subsequent `givePiece()` signals; this process is efficiently driven at the sensor's rate of data production.

Where do the buffers come from? In order to support efficient resource allocation, the developer decides which attributes, if any, share buffers and other resources when implementing the call:

```
command result_t
giveResources (Resources_t *resources);
```

The developer merely allocates the data structure for resources. Sdlib handles all aspects of using the structure.

IV. Next Steps

The current design readily supports GGB, VID and FAB, in addition to its base, Nucleus. We plan to iterate on sdlib to support increasingly disparate applications: TinyDB, SD strongly suggest a need for user-designed in-network data operators, much like the currently supported user-designed attributes. Deluge's bulk data transfer requires applications receive, as well as send large data objects. Many of the applications could greatly benefit from a unified yet efficient addressing, naming and routing scheme.

Acknowledgment

The authors thank the members of the sdlib group for invaluable dialog: Cheng Tien Ee, Joe Hellerstein, Phil Levis, Sam Madden, and Gilman Tolle.

References

[1] S. Madden and M. Franklin and J. Hellerstein and W. Hong, "The design of an acquisitional query processor for sensor networks," Proc. ACM SIGMOD 2003, June 2003, San Diego, California.

[2] B. Greenstein, E. Kohler, and D. Estrin, "A Sensor Network Application Construction Kit (SNACK)," Proc. ACM SenSys 2004, November 2004, Baltimore, Maryland.

[3] P. Levis, D. Gay and D. Culler, "Active sensor networks," Proc. ACM/USENIX NSDI 2005, March 2005, Boston, Massachusetts

[4] G. Tolle, and D. Culler, "Design of an Application-Cooperative Management System for Wireless Sensor Networks," Proc. EWSN 2005, January 2005, Istanbull, Turkey.

[5] D. Chu, "Listing and further description of studied applications," www.cs.berkeley.edu/~davidchu/sdlib/studiedapps.