A Dynamic Programming Framework for Pervasive Computing Environments

Christine Julien
Department of Electrical and Computer Engineering
The University of Texas at Austin
c.julien@mail.utexas.edu

Joachim Hammer
Department of Computer and Information Science and Engineering
University of Florida
jhammer@cise.ufl.edu

William J. O'Brien
Department of Civil Engineering
The University of Texas at Austin
wjob@mail.utexas.edu

ABSTRACT

As lightweight sensors and mobile computing devices become ubiquitous, they provide increasing amounts of context information that can be leveraged for a range of uses. This paper presents a novel architecture for managing the resulting context-aware interactions to provide transparent, lightweight decision support for a variety of domain specific applications where the types, locations, and users of pervasive sensors and devices are dynamic and unpredictable. Our approach takes the current notion of sensor networks from a unified collecting apparatus to a fluid and information rich environment that is directly accessed by mobile users that move in an information-rich environment. Instead of simply dealing with the environment's inherent heterogeneity, we take advantage of the differences among devices to appropriately distribute application functionality throughout the network. While our architecture and its approach to providing context-aware decision support are highly generalizable, we motivate the development of the architecture with the vision of an intelligent construction site.

1. INTRODUCTION

Several technological trends are driving the evolution of ubiquitous computing environments that are quickly becoming integral to existing application domains. These recent novel developments include the miniaturization of powerful computing devices into highly portable laptops, personal digital assistants (PDAs), and even computationally sophisticated cellular phones and pagers. In conjunction with these computing devices, tiny, low-cost sensing devices are beginning to deliver vast amounts of environmental information to networked applications. These developments offer the promise of enabling sophisticated context-aware applications that allow users to directly access context information collected within their local environment.

As computing capabilities and application possibilities expand, the demand for domain specific applications also increases, placing a heightened burden on application programmers. In addition, the time required to communicate sophisticated context and domain dependent requirements to expert programmers dominates the development task. In response, the work reported in this paper creates a customizable framework that simplifies the development of domain-specific applications. The result abstracts the programming task into a simple, usable, yet expressive architecture for building sophisticated real-time decision support applications that leverage context information collected from distributed sensor networks. While the architectural approach is generalizable to many application domains that fit our computational model, we use the intelligent job site to motivate the design and implementation of our architecture.

This work diverges from traditional approaches that commonly treat sensor networks as static arrays for information collection where the entire network has a common global task, and the devices work together to achieve this task. Traditionally, sensing devices forward sensed information back to a single central collection point where the data is used. While some processing often occurs in the network, the goal is usually filtering and aggregation to improve efficiency. We focus on a significantly altered computational model in which environmental information is also accessed in real time through local interactions between mobile devices carried by people in the immediate area.

To achieve the above goals, we have developed a three-tiered architecture that separates decision-support from data and sensor management. From the application programmer's perspective, our approach provides complete transparency of the computational complexity of the environment by allowing the application deployer to choose configurable decision-support modules that perform well-defined tasks. This enables domain experts *with minimal programming expertise* to rapidly construct, deploy, and alter highly adaptive applications in heterogeneous sensor pervasive computing environments.

2. A CASE STUDY: THE INTELLIGENT JOB SITE

The "intelligent job site" [4] as envisioned for construction presents considerable challenges to pervasive computing, requiring both context-awareness and multiple decision support applications for physically distributed (and mobile) devices. The intelligent job site in Figure 1 shows users, mobile devices, and sensors connected by a wireless network. A job site office in the upper left has access to archival data and enables centralized decision-support applications (such as overall materials and trade coordination). Worker B carries a portable computing device that can interact with distributed sensors. For example, worker B can read an RFID tag with materials information (about the palette of bricks) and record its location. Outside the building, worker A nears a VOC (Volatile Organic Compound) source; worker A's

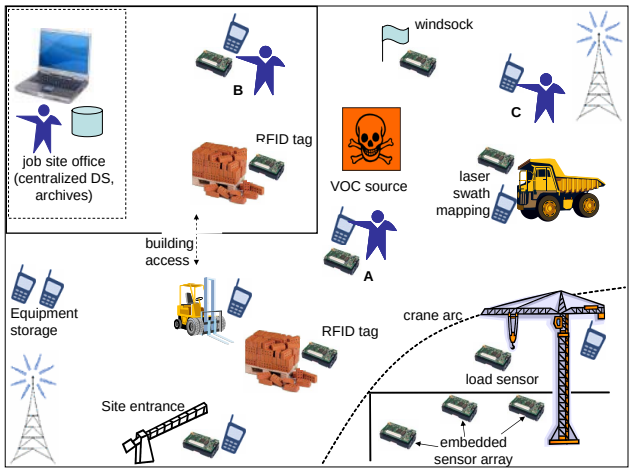

Figure 1: An intelligent construction site.

sensor can monitor the level of the hazardous material and use the network to warn others if the concentration surpasses a threshold. A wind-sock can provide wind information to augment this notification.

Also depicted is a dump truck with a laser swath mapping device used for collision detection. Other onboard devices provide the truck's speed, turning radius, etc. A similar application is shown in the lower right, where a crane has a load sensor that checks if the crane is in danger of exceeding its capacity. A computing device in the cab can also interact with devices on workers and vehicles to check if they are within the arc of the crane, significantly aiding safety maintenance. Also near the crane is an embedded array of sensors for structural health monitoring (these could be heat sensors that measure concrete cure rates or stress/strain sensors in walls or floors). Finally, the lower left shows a site entrance with various RFID readers to record materials deliveries (and on large sites, to provide directions to trucks as they enter the site). The entrance can also provide security, track worker entry and monitor tools and equipment.

Abstracting away from specific applications, it is clear that the intelligent job site presents a wide range of mobile devices that comprise a highly heterogeneous and dynamic sensor network. In this sense, context-awareness is important as, for example, workers on the site enter different zones of potential danger. Further abstractions can be made concerning decision support applications. There are two general classes of applications: 1) near real-time, local decision support and 2) centralized, archival decision support. Local decision support is best evinced by safety applications, such as collision detection, hazardous materials detection, etc. These applications monitor local conditions for a particular device, worker, or vehicle. Computing devices may need to exchange information, and all interactions need to be as local as possible with minimal reliance on potentially unavailable central computing resources. In contrast, centralized archival applications make more extensive use of centralized decision support and require frequent, although not real time, communication with a centralized resource. Materials handling and trade coordination is one example, where there are many degrees of freedom and some level of centralized coordination across the site can lead to improved planning. Challenges involve separation of local and centralized decision support functions, contextual decisions about data to archive on local devices, and efficient reuse of data collected for other applications.

3. THREE-TIER ARCHITECTURE

As sensors become economically viable and pervasive, scenarios like the one described above are becoming commonplace. The radically changing constraints in combination with the required application functionalities demand novel approaches. The requirements of these environments place a strong focus on achieving a lightweight software footprint, minimizing communication and power costs, and deploying adaptive and responsive software. These concerns drive our creation of a flexible architecture for developing and deploying heterogeneous pervasive applications. Figure 2 shows an overview of our architecture, which abstracts functionality into three layers: a low-level layer for communication (the Sensor Communication Layer, or SCL), a layer for data processing (the Data Processing Layer, or DPL), and the uppermost, application layer for decision support (the Decision Support Layer, or DSL). We discuss the details of the architecture and these layers in Section 3.2. We first describe the intended operating environment in a generic sense and the requirements that the computational environment imposes on a software system.

3.1 Computational Environment and System Requirements

First and foremost, we aim to ease interaction with data in information rich environments. To that end, it is crucial that we provide an interface that is customizable, expressive, and flexible. The architecture must also be intuitive to non-programmers (e.g., the supervisor on an intelligent construction site). We do not aim to create customized instantiations of the architecture for every possible domain. On the other hand, due to the complexity of this task, it cannot be left to the users. Our customization interface allows a novice programmer (likely with domain expertise) to

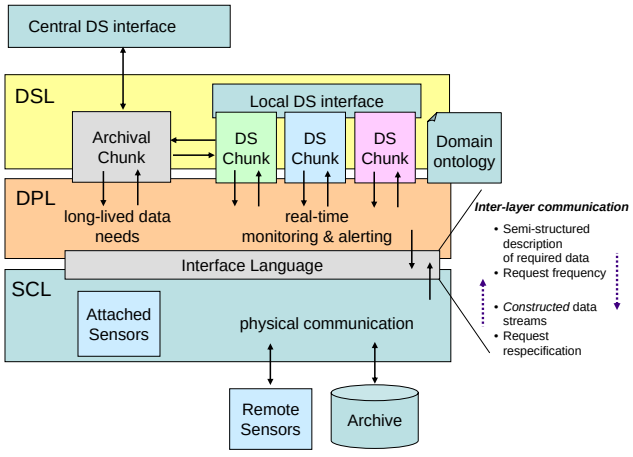

Figure 2: Architecture for lightweight decision support in heterogeneous networks.

tailor the framework to a particular domain or application's needs. Our second goal is to drastically simplify the tasks of this programmer by abstracting the information rich environment into a familiar and usable interaction system.

Existing work in deployed sensor networks [15, 17] focuses on assigning sensors specific, cooperative tasks that collect results at a centralized location. Much research has focused on low-energy information collection [9] and local data aggregation [14]. As sensor networks have become more pervasive, research has targeted the simplification of programming [7, 16] by generating primitives that can be combined into sophisticated applications. Our architecture must remain general, but domain programmers will be required to customize the framework to their specific needs. We believe that our approach provides a much needed tradeoff between the demand for domain specific solutions and the desire to provide a highly general system.

Pervasive environments are characterized by high degrees of mobility, dynamics, and unpredictability. As the case study demonstrates, we not only want to collect information at a centralized location, but many applications also demand that devices in the field interact directly with context sensors to collect fresh data. Our framework has the following concrete goals and requirements:

- *Ease of use.* From a software engineering perspective, the framework must be easy to use for both the domain programmer and the domain user.
- *Generalizability.* The architecture must apply in a variety of domains. To remain general, we encapsulate as much functionality as possible.
- *Customizability.* In competition with the previous goal, many application domains require tailored solutions, so our architecture must have clear points of customization that can provide individualized functionality.
- *Lightweight footprint.* Our architecture must operate on resource constrained devices which impose power, communication, and computational constraints.

3.2 The Architecture Design

Our approach abstracts distinct functionalities into three layers (as shown in Figure 2), which allows users to easily select, deploy, and interact only with the necessary components on their devices.

3.2.1 The Decision Support Layer (DSL)

At the highest level lie the domain specific decision support application components, which make up the Decision Support Layer (DSL). DSL functionality is packaged into *chunks*, which perform well-defined decision-support tasks such as monitoring the local wind speed and alerting its user when the measured wind speed exceeds a threshold. Chunks are developed by domain experts as shared modules for use and customization by others. Not shown in the figure is a chunk repository, which holds chunks along with their meta data (e.g., what data/results the chunk produces, what input it needs, what type of hardware capabilities it requires).

Chunks are divided into two categories: local, near real-time decision support that function autonomously for a particular user (*DS Chunks*), and archival decision support that provide longer-lived information gathering with a more global (e.g., job site) perspective (*Archival Chunks*). DS chunks encapsulate tasks that require access to localized context data or which require near instant response to a specific event. The result of a DS chunk execution can be a sequence of time-dependent values displayed in a local DS interface or an alert that may trigger a follow-up action (e.g., an SMS message sent to another device).

Archival chunks collect and filter data but defer complex processing for when the data is uploaded to the archive. Uploading can be done whenever the archive is reachable from the device via WLAN (i.e., weakly connected mode) or by physically connecting the device to the archive, for example, at the end of a shift (i.e., strongly connected mode). To view an archival chunk's output, a user accesses the DS interface attached to the archive. Here, more complex interactions with the accumulated data are possible including OLAP (Online Analytical Processing) operations such as drill-down and roll-up (see [3] for a general introduction to OLAP). These complex operations are provided by the decision-support system managing the archive and not the archival chunk.

Our approach to chunk development is a high-level specification language that allows non-computer scientists to create new DS chunks and combine existing chunks into aggregates. Our XPDL-based language (XML Process Definition Language [2]) accommodates a range of data (given that sensor data may be incomplete or inaccurate), incorporates contextual data about the environment, accommodates real time data, and has basic aggregation capabilities. Chunks are relatively simple modules containing monitoring and alerting on a limited number of variables. In the future, we will add a high-level work-flow system that will allow simple chunks to be assembled into composite chunks, which can monitor multiple sensors and aggregate their results. For example, a complex decision-support chunk may be able to detect the spread of a noxious gas cloud which may require the simultaneous monitoring of weather conditions, gas leaks, and locations of nearby humans.

3.2.2 The Data Processing Layer (DPL)

Chunks execute in the Data Processing Layer (DPL). The chunk execution engine processes requests encoded in chunks and formulates them into logical query plans. A query plan contains descriptions of the context data needed as well as operations (e.g., selection and filtering) to be applied to the data. In the future we may also add instructions about optimizations such as piggybacking operations onto existing execution threads supporting different chunks. This will reduce the number of communication requests to SCL and hence minimize network traffic and energy consumption.

To ensure that chunks can be executed within the small footprint of the mobile device, queries that involve prolonged monitoring are off-loaded to the archive (i.e., the DS chunk becomes an archival chunk). The decision to off-load is made before the chunk executes; in future work, this may become a dynamic decision. Our approach to data processing leverages existing results, for example, in the areas of stream-processing [5, 12], continual queries [13], and adaptive query processing [8, 10].

Chunks interact with the DPL through an application specific ontology, encapsulated as a plug-in to the architecture, as shown on the right of Figure 2. In the DSL, the ontology customizes the (relatively general) decision-support

language to the application domain. The DPL uses the ontology to aid in discovery and interpretation of data sources available from the SCL, as well as in query planning. Separating the domain ontology also provides a simplified and editable framework for site managers to customize applications to site specific needs. The ontology also registers devices as they enter the application, avoiding expensive discovery at the sensor communication layer.

3.2.3 The Sensor Communication Layer (SCL)

The Sensor Communication Layer (SCL) interacts with the information-rich environment to provide the data required by lightweight decision support applications. Given the abstraction between the DPL and the SCL, communication between the two layers requires care. The DPL's execution engine is unaware of physical sensors. It instead formulates requests in terms of abstract descriptions of data (e.g., a request by a worker on a job site may be for concentrations of hazardous gases in a particular region of the job site). The request also describes the type of result expected (e.g., a stream or single value) and the frequency of the desired readings (for a continuous stream) or the size of the result window (for a snapshot). Specifically, the interactions between the DPL and the SCL occur through a generic interface language based on semi-structured data [1] for describing the attributes and values of the desired data. We augment this scheme with specifications of additional constraints on the streams (e.g., type, frequency, freshness, etc.).

In turn, the SCL takes a request from the DPL and discovers and communicates with the sensor sources to generate the requested streams in a domain-independent manner. In our initial instantiation of the architecture, the SCL uses Cross-layer Discovery and Routing (CDR) [11] for communication among heterogeneous devices. Future work includes incorporating a sensor-specific communication protocol that can take advantage of the domain-specific ontology to decrease the communication overhead associated with sensor discovery. CDR employs attribute-based routing using the semi-structured data specifications to discover satisfying sensors (e.g., a CDR request for hazardous gas data may return a network connection to a VOC sensor). These connections are then leveraged to provide data satisfying the DPL request's constraints. The DPL request may be satisfied by a single physical device or some combination of devices, and the SCL integrates the physically sensed data to match the DPL's data requests. For example, as a worker moves through a job site, the DPL's connection to a continuous stream of VOC data may be supplied by different physical VOC sensors depending on the worker's physical location.

4. DISCUSSION

The separation of the architecture's functionality into three layers allows us to deploy only the necessary functionality on each device while supporting a range of applications over a distributed and heterogeneous network. As shown in Figure 3, a single device or sensor can support one, two, or all three layers depending on its intended use and capabilities. For example, the components of the job site with significant computing and communication power (e.g., a PDA or a computer attached to the crane) support all three layers of the architecture. These devices are most likely to interface

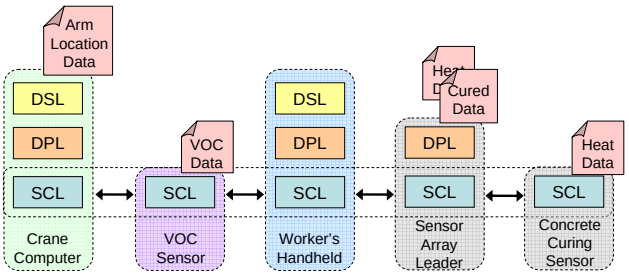

Figure 3: Partial deployment of the three-tiered architecture on devices with varying resources.

directly with users, and they also interact with locally available context sensors on the user's behalf. Other resource constrained sensors (e.g., a VOC sensor) support only the SCL to enable communication.

The multiple layers of abstraction also provide great flexibility in developing decision support applications. Consider a safety application for the intelligent job site. Such an application requires continuous monitoring of the local environment by individual workers' devices. When a condition (such as a VOC sensor measuring a high degree of a hazardous gas) is sensed within the deployed decision support architecture, the SCL is immediately involved on the triggering sensor. The SCL coordinates with the DPL (either directly on the sensor if it supports the DPL or through the network to an SCL on another, more powerful device). After performing stream processing and data aggregation (if necessary), the DPL triggers domain-specific (or even site-specific) DS chunks to compute hazards, request additional readings (e.g., wind direction), etc. The DS chunk's behavior is triggered only when an anomalous condition is detected. In other cases, a DS chunk may execute continuously. A DS chunk for collision detection for the crane in Figure 1 may run whenever the crane is moving. Through the DPL, the DS chunk continuously monitors the movement of workers in a local zone surrounding the crane (indicated by a dashed line in Figure 1). The request the DS chunk makes on the DPL in turns triggers the DPL to continuously monitor streams of sensor data collected by the SCL. Should the potential for a collision exceed a threshold, the DS chunk on the piece of equipment can notify the devices carried by the workers in the region.

In Section 2, we also discussed the need to support more traditional centralized or archival decision support. In such scenarios, sensor data is collected from the network and offloaded to a central infrastructure for further processing, much as in [17]. Using our architecture, we can support such applications and also enable in-network processing by leveraging the availability of distributed DS chunks. Archival chunks deployed on mobile devices can monitor data used by local DS chunks (e.g., an archival chunk for materials and personnel tracking can use the location information collected by the participants in collision avoidance application). The archival chunk decides what data to store locally for later communication with a central server.

As can be seen from the descriptions of a handful of applications, their behaviors involve communication across the three layers, and between DS chunks both within and across

devices. Due to the manner in which it has abstracted and encapsulated related functionalities, our architecture is well suited to support programming such applications. Domain expertise is required for development of DS chucks, and the abstraction of the DS chunk and its associated definition language provide a simplified programming interface to the domain expert. In addition, many of the complexities associated with programming distributed applications are encapsulated within the architecture and handled implicitly on behalf of the programmer.

Of course, there are limitations to our architecture. In contrast to many data driven decision-support applications [6], our DS chunks do not support the full range of aggregation and drill-down that can be done on data cubes, for example. On the other hand, chunks conduct decision-support over non-traditional sources such as mobile sensors. Furthermore, DS chunks and the supporting architecture operate on devices with limited hardware and computing capabilities. As such, we are revisiting many of the assumptions made by existing approaches about the availability of disk space, memory, and connectivity.

5. CONCLUSIONS

This paper has presented an architecture specifically designed to support programming applications for highly heterogeneous pervasive environments. We specifically target environments characterized by large numbers of resource-constrained sensing devices, mobile computing devices, and their need to interface with each other and with a centralized archive for a variety of decision support tasks. The founding premise of this architecture is that specific application domains increasingly require tailored application solutions (like those described for the intelligent construction site), but lack programming experts with sufficient domain expertise to develop the applications. To that end, we have developed an architecture that hides the complex details of programming sophisticated context-aware applications. The architecture enables a novice programmer with domain expertise to tailor the framework by creating domain-specific decision support chunks and a domain- or application-specific ontology. The benefits of using this architecture lie not only in its ability to simplify the programming task through the use of encapsulation and abstraction but also in its novel use of partial deployment on resource constrained devices. This allows the architecture to function as a seamless system even in the face of significant degrees of heterogeneity, mobility, and unpredictability.

6. REFERENCES

[1] S. Abiteboul. Querying semi-structured data. In *Proc. of the 6^{th} Int'l. Conf. on Database Theory*, pages 1–18, January 1997.

[2] W. M. Coalition. Workflow process definition interface – xml process definition language, version 1.0. Technical Report WFMC C-1025, The Workflow Management Coalition, 2002.

[3] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. Technical report, Hyperion Solutions Corporation, August 1993 1993.

[4] Fiatech. Strategic overview: Capital projects technology roadmap initiative (version 1). `http://www.fiatech.org/projects/roadmap/cptri.htm`, 2003.

[5] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc. of the Int'l, Conf. on Management of Data*, pages 13–24, 2001.

[6] J. Gray, A. Bosworth, A. Laymen, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proc. of the 12^{th} Computer Science Int'l. Conf. on Database Engineering*, pages 152–159, 1996.

[7] B. Greenstein, E. Kohler, and D. Estrin. A sensor network application construction kit (SNACK). In *Proc. of the 2^{nd} Int'l. Conf. on Embedded Networked Sensor Systems (SensSys'04)*, pages 69–80, 2004.

[8] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, 2000.

[9] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. on Networking*, 11(1):2–16, February 2003.

[10] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proc. of the Int'l. Conf. on Management of Data*, pages 299–310, 1999.

[11] C. Julien and M. Venkataraman. Resource-directed discovery and routing in mobile ad hoc networks. Technical Report TR-UTEDGE-2005-01, University of Texas, 2005. (submitted for publication).

[12] S.-J. Lee, C.-J. Lee, Y. Z. Cho, and S. U. Kim. A new data aggregation algorithm for clustering distributed nodes in sensor networks. In *Proc. of the 3^{rd} European Conf. on Universal Multiservice Networks*, volume 3262 of *LNCS*, pages 508–520, 2004.

[13] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering - Special Issue on Web Technology (OpenCQ)*, 1999.

[14] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, Winter 2002.

[15] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. of SenSys*, pages 214–226, 2004.

[16] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proc. of the 1^{st} Symp. on Networked Systems Design and Implementation (NSDI'04)*, pages 29–42, 2004.

[17] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proc. of SenSys*, pages 13–24, 2004.