SmartCIS: Integrating Digital and Physical Environments[*]

Mengmeng Liu Svilen R. Mihaylov Zhuowei Bao Marie Jacob
Zachary G. Ives Boon Thau Loo Sudipto Guha

Computer & Information Science Department, University of Pennsylvania, Philadelphia, PA, USA
{mengmeng,svilen,zhuowei,majacob,zives,boonloo,sudipto}@cis.upenn.edu

ABSTRACT

With the increasing adoption of networked sensors, a new class of applications is emerging that combines data from the "digital world" with real-time sensor readings, in order to intelligently manage physical environments and systems (e.g., "smart" buildings, power grids, data centers). This leads to new challenges in providing programmability, performance, *extensibility*, and *multi-purpose* heterogeneous data acquisition. The ASPEN project addresses these challenges by extending data integration techniques to the distributed stream world, and adding new abstractions for physical phenomena. We describe the architecture and implementation of our ASPEN system and its showcase intelligent building application, SmartCIS, which was demonstrated at SIGMOD 2009. We summarize the new query processing algorithms we have developed for integrating highly distributed stream data sources, both in low-power sensor devices and traditional PCs and servers; describe query optimization techniques for federations of stream processors; and detail new capabilities such as incremental maintenance of recursive views. Our algorithms and techniques generalize across a wide range of data from RFID and light measurements to real-time machine usage monitoring, energy consumption and recursive query computation.

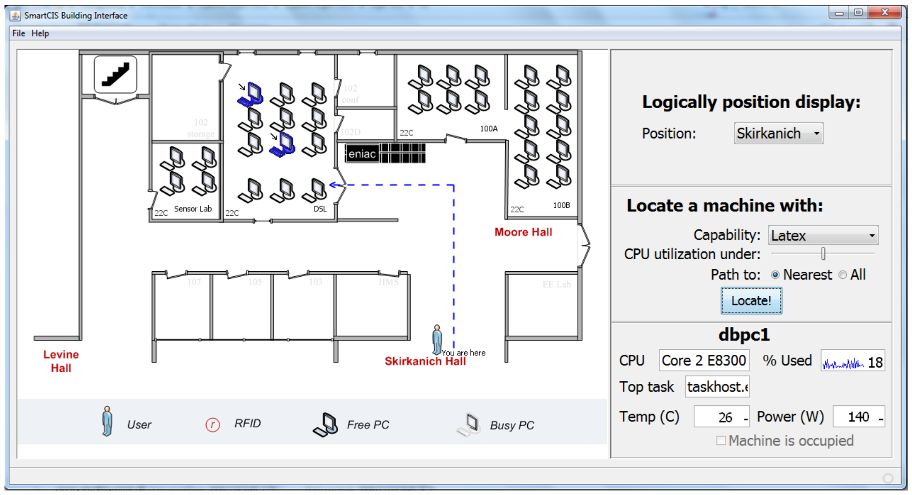
1. INTRODUCTION

Low-cost networked sensors are resulting in a new class of applications that combine data from the "digital world" with sensor readings, to create environments that intelligently manage resources and assist humans. Examples include intelligent power grids [19], smart hospitals [18], home health monitors, energy-efficient data centers, and building visitor guides. In such applications, there is a need to bring together disparate data from databases (e.g., site information, patient treatments, maps) with data from the Web (e.g., weather forecasts, calendars), from streaming data sources (e.g., resource consumption within a server), and from sensors embedded within an environment (e.g., generator temperature, RFID readings, energy levels) — in order to support decision making by high-level application logic. Today this sort of data integration, if done at all, is performed by a proprietary software stack over fixed devices.

In order for intelligent environments to reach their potential, what is necessary is an *extensible*, *multi-purpose* data acquisition and integration substrate through which the application can acquire data — without having to be coded with special support for new device or network types Over the past 30 years, the database community has developed a wealth of techniques for performing data integration through views and related formalisms [11]. Likewise, declarative queries have been shown to be useful beyond databases, with extensions for distributed data stream management [2, 3, 4, 9] and sensor networks [5, 6, 14]. The key question is how to develop a unified declarative query and integration substrate, which supports a multitude of stream and static data sources on heterogeneous, possibly unreliable networks. Computation should be expressed in a single query language and "pushed" to where it is most appropriate, taking into account capabilities, battery life, rates of change, and network bandwidth.

The ASPEN (Abstraction-based Sensor Programming ENvironment) project tackles these issues, extending the formalisms of data integration (schema mappings, views, queries) to the distributed stream world. We are developing (1) new query processing algorithms suitable for integrating highly distributed stream data sources, both in low-power sensor devices [15, 16] and more traditional PCs and servers [13], (2) query optimization techniques for federations of stream processors specialized for sensor, wide area, and LAN settings, and (3) new datatypes, query extensions, and data description language abstractions for environmental monitoring and for routing information to users. In support of smart environments, we seek a *single data access layer* for integrating sensor, stream, and database data, regardless of origins. This single programming interface over heterogeneous sensors and stream sources distinguishes us

[*]This work was funded by NSF III IIS-0713267, NOSS CNS-0721541, and a grant from Lockheed Martin.



Logically position display:

Position: Skirkanich

Locate a machine with:

Capability: Latex

CPU utilization under:

Path to: [✓ Nearest] All

Locate!

dbpc1

CPU Core 2 E8300 % Used 18

Top task taskhost.

Temp (C) 26 Power (W) 140

☐ Machine is occupied

Figure 1: Display indicating a path to, and information about, the nearest machine with LaTeX.

from other sensor systems [7, 10, 14].

The showcase application for the ASPEN architecture, which we term SmartCIS, involves instrumenting Penn's Computer and Information Science (CIS) Department buildings, labs, and data centers to help improve energy efficiency, guide visitors to their desired destinations, and locate resources. Our live demonstration of SmartCIS [12] at SIGMOD 2009 received Honorable Mention for Best Demo. SmartCIS consists of GUI and query logic built over the ASPEN data integration substrate. It combines information from on-site sensors (e.g., pressure-sensitive seat cushions, RFID tags, energy meters) with data from the Web (calendars) and from our Distributed Systems Laboratory at Penn (machine and desk-occupied status; machine configurations).

We describe the SmartCIS prototype in Section 2. Then we present the underlying ASPEN system: its architecture (Section 3), federated query optimizer (Section 4), distributed stream query processor (Section 5), and sensor network subsystem (Section 6). We summarize related work in Section 7 and conclude in Section 8.

2. SMARTCIS BUILDING APPLICATION

One of the most compelling emerging applications of sensors are intelligent building environments: they promise to make the experience of visiting a large building or a hospital less disorienting, to make buildings or large data centers more energy-efficient, to help occupants remember to take their medications or make it to a next meeting. A distinguishing feature of such environments, versus other sensor network applications, is a need to bring together database data with streaming data from the Web or Internet and streaming data from sensor devices. The task of designing a smart building can be separated into three tiers: data acquisition and integration, query and control logic, and a user-interface view (analogous to model-view-controller architectures).

The initial version of SmartCIS focuses on monitoring and querying the data of interest to CIS students and faculties, as well as system administrators: lab status, machine activity, resource consumption, and machine physical state. We target two main tasks: giving a real-time update of the building state, and guiding students to the resources they need. Through the SmartCIS GUI, visitors can see occupied and unoccupied desks in the laboratories and on-site (detected through the seat sensors); their positions in the building (obtained via RFID); temperature, light, and energy usage levels for every machine and lab; room reservation status from Google Calendar; and the resources available at each machine (e.g., software, special equipment). Visitors can see status information or issue a query for directions (a physical path) to a machine with a particular resource.

2.1 User Experience

SmartCIS interacts with users through a touch interface on a kiosk or (for the demo) a tablet PC. Figure 1 shows a screen shot of our graphical interface, which centers around a building schematic. In the full application, the user will see the individual information on a kiosk located somewhere in the building. Our screen shot shows the demo application, which has a selector in the upper right-hand corner enabling a SIGMOD attendee to choose a simulated kiosk location.

Buildings, entrances and exits, rooms, and machines are illustrated schematically. Their status is refreshed in real-time based on data streams from the environment and the Web, combined with database information about locations and configurations. Rooms are grayed out when marked as reserved in a standard Google calendar, or when their lights are out (as detected by sensors). Machines are grayed out when they are currently in use (as detected by high CPU utilization or a pressure-sensitive seat cushion connected to a Crossbow iMote). The presence of a user is detected through active RFID tags (IRIS motes that broadcast a low-power signal that is tracked by stationary motes located throughout the building hallways) and is indicated in the schematic.

The user can also trigger new continuous queries over the streaming data in the system. Clicking on a machine icon switches the right-hand pane to show details about that device: its host name (from a database table mapping coordinates to machine identities), CPU type (also from the database), CPU utilization and the most CPU-intensive task (from a "soft sensor" application), temperature (from an iMote), and energy (from a USB energy meter or an IP-based Power Distribution Unit or PDU). A double-click opens up a secondary window showing energy consumption on a per-task basis (scaling overall energy consumption by the amount of resources consumed by each process). Finally, a visitor can also request to be directed to an available machine with specific resources (e.g., software packages like Microsoft Office or a video editor). A shortest-path query is initiated between the user's current location and the nearest available room with the specified resource.

2.2 Sensors and Data Sources

The data sources underpinning SmartCIS are heterogeneous, requiring a variety of *wrappers* (interface modules), and can be divided into four broad categories.

Sensor devices. We use Crossbow IRIS and iMote2 sensors to monitor the rooms' and workstations' temperatures, as well as light levels (useful for determining if a lab is open). A pressure-sensitive seat cushion attached to a wireless mote monitors whether someone is seated at each desk in the lab. A "wrapper" periodically extracts this value and sends it along a data stream. Energy meters are physically plugged into machines and feed raw readings into the system. To track users' locations, "mote" sensors are embedded in the hallways at major intersection points, at approximately every 50 feet. These sensors listen for a "beacon" transmission from an active RFID device (also a mote) carried by an occupant and based on the strength of the signal determine where that person is positioned in the building.

"Soft" sensors. Servers and workstations run daemon software to monitor machine activity: jobs executing, users logged in, CPU utilization, number of requests being handled in a Web server application, etc. In addition, the status of ASPEN, our back-end data acquisition and integration substrate itself, is also monitored: the queries and plans being executed, the counts of tuples received and sent for every operator, etc. This helps developers diagnose problems at the query execution level and also helps determine per-query energy usage.

Web and streaming data sources. A wrapper periodically polls a Google Calendar for room reservations. Another wrapper polls energy usage from a Web interface to our lab's power distribution units (PDUs).

Databases. A conventional DBMS stores the coordinates of each RFID detector (the motes have no built-in absolute positioning capability), a list of machine configurations and locations, and a table of "routing points" describing possible path segments and distances in the building in order to suggest routes to resources.

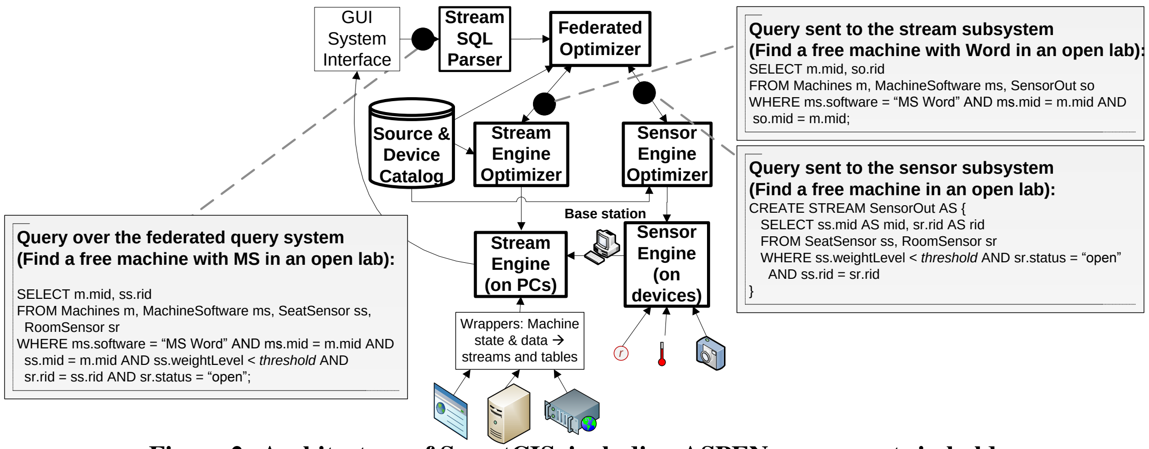
The data from these inputs is "hooked" to the SmartCIS GUI through a series of Stream SQL queries and view definitions, plus callbacks to Java functions that update the graphical widgets. It is trivial to extend the GUI to support visual or auditory alarms if machines exceed a temperature or load factor, or to aggregate the sensor data across users, applications, or machines. Even the path routing in the GUI is done declaratively, using recursive extensions to Stream SQL. We next describe how SmartCIS maps onto the ASPEN substrate that provides distributed Stream SQL services.

3. SYSTEM ARCHITECTURE

The SmartCIS system consists of three major components: the graphical interface described previously, which can be deployed on kiosks; the ASPEN data integration and acquisition substrate, which includes two query runtime systems (one that enables certain computations to be "pushed" to sensor devices, and one that does distributed stream processing over PC-style servers) plus a federated query optimizer; and wrappers and interfaces over the actual sensors, databases, and machines. (See Figure 2.) Components of the ASPEN substrate appear in boldface. (Ultimately ASPEN will also include support for schema mappings and query reformulation, but SmartCIS does not require these components.)

Most of the research innovations are in the ASPEN modules. ASPEN takes a query (Stream SQL with extensions for devices and for routing query output to displays) and invokes a federated query optimizer that partitions it into two portions (see Figure 2): a subquery that is "pushed" out to the sensor network and sensor devices, and the remaining computations that get executed on our distributed stream engine for servers.

The distributed sensor engine, whose core features were described in [15], is novel in supporting not only aggregation and selection queries over sensor devices, but also *in-network* joins between devices. This is useful in SmartCIS, for instance, when we return machine temperature data for workstations that are in use. We detect that a workstation is being used by checking the status of the seat cushion as well as the light level at an adjacent chair. The most efficient query strategy is to perform a proximity-based join between status of seat cushion and light level sensors (with a threshold applied on the light level), and route the temperature information across the sensor network only if the light level threshold is not met. A query optimizer decides where to perform the join computation on a sensor-by-sensor basis.



Query sent to the stream subsystem (Find a free machine with Word in an open lab):

```
SELECT m.mid, so.rid
FROM Machines m, MachineSoftware ms, SensorOut so
WHERE ms.software = “MS Word” AND ms.mid = m.mid AND
 so.mid = m.mid;
```

Query sent to the sensor subsystem (Find a free machine in an open lab):

```
CREATE STREAM SensorOut AS {
  SELECT ss.mid AS mid, sr.rid AS rid
  FROM SeatSensor ss, RoomSensor sr
  WHERE ss.weightLevel < threshold AND sr.status = “open”
   AND ss.rid = sr.rid
}
```

Query over the federated query system (Find a free machine with MS in an open lab):

```
SELECT m.mid, ss.rid
FROM Machines m, MachineSoftware ms, SeatSensor ss,
  RoomSensor sr
WHERE ms.software = “MS Word” AND ms.mid = m.mid AND
  ss.mid = m.mid AND ss.weightLevel < threshold AND
  sr.rid = ss.rid AND sr.status = “open”;
```

Figure 2: Architecture of SmartCIS, including ASPEN components in bold.

Our distributed stream engine, described in [13, 20], supports not only Stream SQL queries over windowed data, but also *transitive closure queries* to compute neighborhoods and paths. The stream engine brings together streaming data, database data, and the data returned by the subqueries sent to the sensor engine. It is also responsible for computing suggested routes for building occupants to get to their destination: this can be done in real-time based on the occupant's current position and information about the topology of the buildings (connected by routing points described previously).

4. FEDERATED OPTIMIZER

ASPEN's federated query optimizer assigns an incoming query across multiple subsystems, each of which has its own custom optimizer and cost metric, customized to the target device and network capabilities (e.g., energy, latency, bandwidth). We give an example of the federated optimizer's optimization in SmartCIS.

Suppose we have two types of sensors deployed in the lab, seat sensors and room sensors. Each seat sensor is pre-initialized with information about its position relative to a machine and the room; it reports the occupied-status of the seat cushion to which it is attached. Each room sensor is pre-initialized with its room, and detects the current light level to tell whether the room is occupied or not. Suppose we also have a static table $Machines$ storing machine information for the lab, and a dynamic stream $MachineSoftware$ containing information about installed software and versions from a web page. The user may pose a query to find all the free machines in an open lab which have "Word."

```
SELECT m.mid, sr.rid
FROM Machines m, MachineSoftware ms,
  SeatSensor ss, RoomSensor sr
WHERE software = "Word" AND ms.mid =
  m.mid AND ss.mid = m.mid AND ss.weightLevel
  < threshold AND sr.rid = ss.rid AND
  sr.status = "open";
```

There are multiple plausible ways of splitting the query. One method pushes the `SeatSensor`-`RoomSensor` join and all relevant selection conditions to the sensor subsystem, then sends the output to the stream engine. (Example SQL for this scenario is shown in Figure 2.) Alternatively, we can issue *two* subqueries to the sensor subsystem: one to fetch `SeatSensor` readings above threshold, and the other to fetch `RoomSensor` readings with open status. Intuitively, the first query partitioning is likely to return fewer results to the stream system only if the predicates are selective.

The federated optimizer must choose among these and other plans by minimizing an over-arching cost metric (e.g., query latency). This metric may be *different* from the metrics of the "local" optimizers for the underlying stream and sensor engines (e.g., bandwidth, energy consumption). The federated optimizer must find a query partitioning that, when each subquery is optimized according to its target platform's specific metric, results in the best plan with respect to the federated optimizer's over-arching metric. Its plan enumeration strategy resembles that of [8], which predicts the query plan produced by an external optimizer, in order to produce the minimum-cost plan according to its own metric.

5. STREAM ENGINE

Our stream engine is derived from the distributed SQL processor from ORCHESTRA [20]. This engine supports horizontal partitioning of data across nodes within a cluster or peer-to-peer network, and is based on a push-style query processing model. We enhance the engine with support for continuous queries (where the query is active unless deliberately stopped) over windows, where the size of the sliding window tells the system when to evict expired tuples. The engine can seamlessly combine data from streaming sources, tables partitioned throughout the cluster, ODBC/JDBC sources, and the sensor query engine. The query optimizer uses a Volcano-style top-down dynamic programming algorithm, and takes into account the network latency as well as data transmission rate when estimating the cost of a certain query plan.

A novel aspect of our engine is its support for *re-*

cursive queries (such as shortest paths) computed (and incrementally maintained) over streaming data. Such queries commonly appear in sensor settings. We have developed techniques, documented in [13], based on (1) the use of a particular kind of *data provenance* that enables us to detect when a tuple in the output stream should be expired, (2) early pruning of intermediate results that do not contribute to the output, and (3) careful use of buffering to reduce traffic. SmartCIS exploits these features to compute path queries, when a user requests to be directed to a resource within the building.

6. SENSOR ENGINE

The sensor engine collects environmental data, potentially from several independent sensor networks. Each sensor network deployment consists of wireless devices situated within the observed environment, and a gateway node to the core of the ASPEN system. One of the sensor devices is connected via USB to the gateway node, serving as a *base station* for the rest of the wireless network. We do not assume that all wireless nodes remain in the radio range of the base station; we focus on effectively utilizing multi-hop wireless networks.

Sensor query capabilities. Our sensor subsystem supports windowed Stream SQL queries (subqueries sent by the federated optimizer) with arbitrary selection conditions, and optionally a single in-network join. Selection and join predicates can include not only standard comparisons and Boolean operations, but also arithmetic operators and several utility functions (e.g., hash, random value). We model each mote sensor network deployment as a single relation with attributes including sensor values (e.g., temperature, light, humidity, battery level, RFID being detected, ADC values) and *soft-state* readings (e.g., memory available, local time). Not all attributes need to be defined for every node, as our system allows for different device capabilities. We also allow for additional data values to be stored at each device from external tables. The update rate for physical sensors is specified as part of the query, as are other parameters such as query start times, join window size, etc.

Basic operation and coordination. The gateway machine not only bridges between TCP and ZigBee (mote) networks, but also plays a supervisory role in the sensor engine. It supervises the construction and initialization of the wireless network after all wireless devices are turned on (described in the next sub-section). It also collects statistical information for the federated optimizer, such as the number of wireless nodes present, network diameter and distribution of values in the different regions of the network. The gateway also serves as a coordinator for *partitioning and storing* certain tables within the sensor network: often it is useful to take certain database tables (e.g., a mapping between node identity and position or role) and to partition them such that one tuple is stored at each wireless node. Then we can push selection conditions relating to these tables directly into the network, optimizing communication efficiency. Finally, it takes sensor network subqueries from the federated optimizer, and performs a "local" network-specific optimization of those queries for the sensor network.

We now briefly describe initialization, optimization, and query execution; [15, 16] provide more details.

Sensor network initialization. Right after the wireless network is started, the first step is to create a sensor network topology that approximates the connectivity graph among sensors. A set of spanning trees is created, one rooted at the base station node, and others rooted at nodes located at opposite extremities of the network. Each internal node in the trees maintains a *summary* (Bloom filter, histogram, R-tree) of the values for a particular attribute that appear in each subtree. The summaries are used for content-based routing [15].

Query pre-optimization. When a subquery is sent to the gateway machine, a query pre-processor first separates the predicates in the query into selections and joins. Then, predicates from each group are separated into static and dynamic subgroups, depending of their attributes being exclusively static or not. Each static join predicate is further fed into a pattern matcher, which, given a collection of summaries built on various static attributes, decides whether the predicate is suitable for content routing using our substrate. In essence, the pattern matcher identifies those join predicates usable for routing, versus runtime evaluation. Finally, the gateway node considers different ways of distributing the evaluation of expressions. Consider the following query with user-defined functions F and G:

```
SELECT S.u+S.v, F(2*S.u+S.v), G(S.u), G(S.v)
FROM SENSORS S [windowsize=1
   sampleinterval=100]
WHERE S.u > 0 AND S.id = 0;
```

This query asks for four evaluations on attributes for every node in the network satisfying $S.u > 0 \wedge S.id = 0$. If at a given node, the selection condition is satisfied, a trivial execution strategy will compute the four evaluations and send them to the base station node. A more efficient strategy will send only $S.u$ and $S.v$, which the base station can in turn use to compute the four original evaluations. In general the intermediate evaluations can themselves be expressions. The problem is exponential in the number of evaluations, and our implementation uses a combination of dynamic programming and heuristics, which performs optimally for queries we used in our testing and experimentation.

Once the pre-optimization is finished, the query is encoded and flooded to every node in the wireless network.

Distributed optimization and execution. For single-relation queries there is no optimization phase, so query execution proceeds immediately. Otherwise, query optimization is invoked at each node, which checks if its attributes satisfy the static selection conditions. If so it initiates a *directed flood* routing request, searching for nodes that mutually satisfy the static join selection conditions. The directed flooding algorithm uses the summaries constructed during initialization [15]. If such nodes are discovered, their selection conditions are checked, and a *join pair* is established. Consider the query:

```
SELECT S.id, T.id, S.time
FROM SENSORS S, SENSORS T
     [windowsize=3 sampleinterval=100]
WHERE S.id < 25 AND hash(S.u) % 2 = 0
  AND T.id > 50 AND hash(T.u) % 2 = 0
  AND T.y = S.x + 5 AND S.u = T.u
```

First, all nodes (playing the role of S) check if their id is less than 25. If so, they issue a routing request to find nodes for which $T.y = S.x + 5$ and $T.id > 50$. Following the evaluation of the static predicates and establishment of paths between joining nodes, a *join node* is assigned for each join pair. Candidate join nodes are those on the path connecting the source nodes, as well as the base station. The join node is chosen using a cost model, based on the estimated relative selectivities of each relation, and the relative network distances (see [16]).

Query execution samples attributes (humidity, temperature, light levels, ADC voltages) at regular intervals, and evaluates the dynamic selection conditions. If the conditions are satisfied, a tuple containing intermediate evaluations is sent to either the join node (if applicable) or the base station. Each join node collects tuples from both relations and computes the join result by evaluating the dynamic join predicate. The join node also tracks any changes to the relative selectivity of the relations it handles, and may trigger adaptation of the join node placement, as described in [16]. Finally, results are sent to the base station and then to the gateway machine.

7. RELATED WORK

In the past decade several influential distributed stream systems [3, 9, 17] have been proposed. These have established the basic semantics and query languages for stream processing. In parallel, stream SQL techniques have been shown to be highly advantageous in a sensor setting [5, 6, 14]. Work such as REED [1] has shown that there is promise in coupling the two classes of systems. We seek to take this idea further, with a federated model supporting distributed optimization across multiple cooperating stream sub-systems, each tailored to particular device classes; support for data integration capabilities; recursion for path and region queries.

8. CONCLUSIONS

This paper provided a technical overview of the SmartCIS "smart building" application and its underlying ASPEN substrate. We introduced new query processing schemes for integrating highly distributed stream data sources, both for low-power sensor devices and servers, as well as query optimization techniques for federations of stream processors. Future work includes designing a more flexible federated optimizer, adaptive query processing techniques for our highly distributed setting, and support for user-defined functions.

9. REFERENCES

[1] D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *VLDB*, pages 1361–1364, 2004.

[2] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2), 2006.

[3] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the Borealis distributed stream processing system. *ACM Trans. Database Syst.*, 33(1), 2008.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[5] A. J. Demers, J. Gehrke, R. Rajaraman, A. Trigoni, and Y. Yao. The Cougar project: a work-in-progress report. *SIGMOD Record*, 32(3), 2003.

[6] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD*, 2006.

[7] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, 2005.

[8] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *VLDB*, 1997.

[9] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Quering the Internet with PIER. In *VLDB*, 2003.

[10] N. Khoussainova, E. Welbourne, M. Balazinska, G. Borriello, G. Cole, J. Letchner, Y. Li, C. Ré, D. Suciu, and J. Walke. A demonstration of cascadia through a digital diary application. In *SIGMOD*, New York, NY, USA, 2008.

[11] M. Lenzerini. Tutorial - data integration: A theoretical perspective. In *PODS*, 2002.

[12] M. Liu, S. Mihaylov, Z. Bao, M. Jacob, Z. G. Ives, and B. T. Loo. SmartCIS: Integrating digital and physical environments. In *SIGMOD*, 2009.

[13] M. Liu, N. E. Taylor, W. Zhou, Z. G. Ives, and B. T. Loo. Maintaining recursive views of regions and connectivity in networks. *TKDE*, 2010. Special issue on best papers of ICDE 2009.

[14] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[15] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. A substrate for in-network sensor data integration. In *DMSN*, August 2008.

[16] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. Dynamic join optimization in multihop wireless sensor networks. In *Proc VLDB*, 2010. To appear.

[17] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, 2003.

[18] J. V. Sutherland, W.-J. van den Heuvel, T. Ganous, M. M. Burton, and A. Kumar. *Future of Intelligent and Extelligent Health Environment*, volume 118/2005, pages 278–312. IOS Press, 2005.

[19] J. Taft. The intelligent power grid. *Innovating for Transformation: The Energy and Utilities Project*, 6:74–76, 2006. Available from www.utilitiesproject.com.

[20] N. E. Taylor and Z. G. Ives. Reliable storage and querying for collaborative data sharing systems. In *ICDE*, 2010.