

# SmartCIS: Integrating Digital and Physical Environments

Mengmeng Liu, Svilen Mihaylov, Zhuowei Bao, Marie Jacob, Rahul Iyer, Easwaran Subbaraman  
Zachary Ives, Boon Thau Loo, Sudipto Guha

{mengmeng, svilen, zhuowei, majacob, iyerr, easwaran, zives, boonloo, sudipto}@cis.upenn.edu

April 8, 2009

## 1 Introduction

As networked sensors continue to grow in sophistication and decrease in cost, we are seeing a new class of applications: those that combine data from the digital world with sensor readings, to facilitate environments that intelligently manage resources and assist humans. Examples include intelligent power grids [13], smart hospitals [12], home health monitors, energy efficient data centers, and building visitor guides.

In all of these 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 full 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 types, new network types, and new datatypes. Over the past 20 years, the database community has developed a wealth of techniques for performing data integration through queries, views, and related formalisms [8]. Likewise, declarative queries have already been shown to be useful beyond databases, with extensions for distributed data stream management [1, 2, 3, 7] and sensor networks [4, 5, 10]. 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, and network bandwidth.

The ASPEN (Abstraction-based Sensor Programming ENvironment) project tackles these issues, focusing on extending formalisms of data integration — schema mappings, views, queries — to the distributed stream world. Points of emphasis are (1) developing new query execution and optimization algorithms suitable for integrating highly distributed stream data sources, both in low-power sensor devices [11] and more traditional PCs and servers [9], (2) developing query optimization techniques across a federation of stream processors specialized for sensor, wide area, and LAN settings, and (3) developing 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.

To evaluate our work, we have been developing a showcase application: instrumenting our Computer and Information Science buildings, labs, and data centers with devices and user interfaces to improve energy efficiency, guide visitors to their desired destinations, and find free desks and laboratories. Our application, SmartCIS (Smart Computer and Information Science Building), forms the centerpiece of our demonstration.

SmartCIS consists of a suite of sensor devices deployed throughout a portion of Penn's Moore building (which holds most of our laboratories), a set of "soft sensors" (monitors of logical state) running on computers, and a graphical interface and control logic. Functionality is enabled via the ASPEN data acquisition and integration substrate.

## 2 Developing A Smart Building

One of the most compelling emerging applications of sensors is intelligent building environments: they promise to make the experience of visiting a large building or a hospital less disorienting, to make buildings or large datacenters more energy-efficient, to help occupants remember to take their medications or make it to a next meeting. A distinguishing feature of such environments, when compared with 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, control logic, and a user-interface view (analogous to model-view-controller architectures).

As a testbed and showcase of intelligent environments and our ASPEN substrate, we have been developing the SmartCIS application, which monitors occupancy and locations, machine activity, and machine physical state. It also incorporates data from databases and the Web. We target two tasks: monitoring the space that people (students) use in order to guide them to destinations, and monitoring machines in order to facilitate adaptive power management or to detect failures. SmartCIS has the following capabilities:

**Room monitoring.** Laboratories and offices can be monitored for temperature and light on/off status. This can be used, e.g., to determine whether a laboratory is open or a room is occupied.

**Machine-state monitoring.** Servers and workstations run software that monitors machine activity: jobs executing, users logged in, CPU utilization, memory, number of requests being handled in a Web server application. This helps determine machine usage, including patterns *across* machines.

**Workstation monitoring.** Servers and workstations are plugged into power distribution units (PDUs) with Web interfaces showing current power consumption. A "wrapper" periodically (every 10s) extracts this value and sends it along a data stream. IRIS or iMote2 sensors mounted on each machine monitor its temperature. At each desk, the light-level sensor on a similar "mote" is used to detect if someone is seated in the chair.

**Detection of occupants.** "Mote" sensors are embedded in the hallways at major intersection points, and every 100 feet. These sensors listen for a "beacon" transmission from an active RFID device (also a mote) carried by an occupant and determine where that person is positioned in the building.

**Databases and Web sources.** We incorporate database information specifying the coordinates on the map of each RFID detector (the motes have no built-in positioning capability), a list of machine configurations and locations in each laboratory, and a table of "routing points" describing possible path segments and distances in the building in order to suggest routes to resources.

Based on these inputs, SmartCIS supports a variety of queries. We can trigger alarm notifications if machines exceed a temperature or load factor. We can monitor the total resources used (energy, memory, CPU) by any user or application, even across machines. We can find available machines in the laboratories, even by capability. We can determine where a visitor is located. Finally, we can do path routing in the buildings, in order to guide the visitor to a destination. Our graphical displays are located on laptops with wireless access, which may be virtually "mapped" to positions in the building. At each display, the user may select a query by adjusting controls, e.g., double-click on a machine or laboratory to monitor, or via a combo box choose a resource to locate.

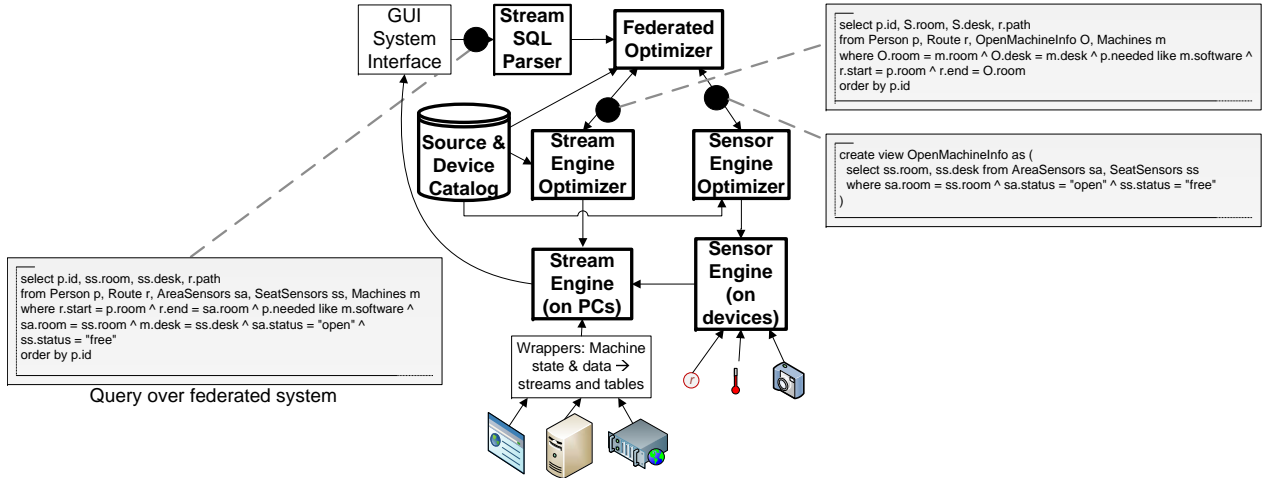


Figure 1: Architecture of SmartCIS, including ASPEN components in bold.

### 3 SmartCIS-ASPEN Architecture

The SmartCIS system architecture consists of three major components: a graphical interface for authoring queries and returning results, which is deployed on machines in the environment; 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 and workstations) plus a federated query optimizer; and wrappers and interfaces over the actual sensors, databases, and machines. (See Figure 1.) Components of the ASPEN substrate are highlighted in boldface. (Ultimately ASPEN will also include support for schema mappings and query reformulation, but for SmartCIS these components are not necessary.)

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 (as in Fig. 1): a subquery that will be “pushed” out to the sensor network and sensor devices, and the remaining computations that will be executed on our distributed stream engine for servers and workstations.

The distributed sensor engine, whose core features were described in [11], is novel in supporting not only aggregation and selection queries over sensor devices, but *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 for a low light-level at the adjacent chair. Hence, the most efficient query strategy is to perform a proximity-based join between temperature and light-level sensors (with a threshold applied on the light level), and to only route temperature data across the sensor network if the light threshold is met. Our sensor engine includes a query optimizer that decides, on a sensor-by-sensor basis, where to perform the join computation.

Our distributed stream engine, described in [9], supports not only basic Stream SQL queries over windowed data, but also *transitive closure queries* that enable computation of neighborhoods and paths. The stream engine performs most of the query processing within SmartCIS, bringing together stream data, database data, and data output 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 (the routing points described previously).

The federated query optimizer supports multiple underlying heterogeneous distributed query engines.

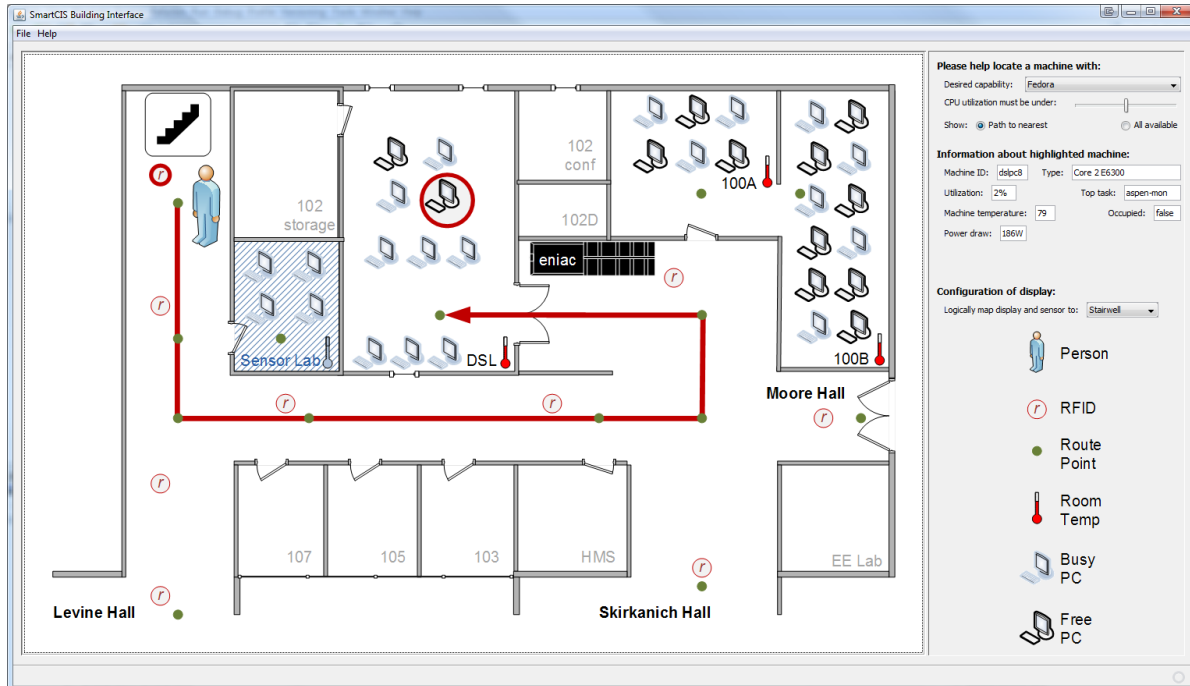


Figure 2: Screenshot of GUI showing building layout, open and closed (shaded with dashed lines) labs, free and unavailable machines, and a path to and details about the nearest machine with Fedora Linux.

Somewhat along the lines of the model established in the Garlic system [6], the federated optimizer enumerates all possible plans, and partitions these plans among the different query engines. It invokes the optimizer for each query engine over its assigned partition, and determines (1) whether this is a query plan the engine can actually execute<sup>1</sup>, and (2) what the cost of the query partition would be. The novelty in ASPEN is that the cost models of the different sub-optimizers may return different cost parameters: the sensor optimizer attempts to minimize message traffic, whereas the stream optimizer attempts to minimize latency to answers. The federated optimizer must convert everything to one model, in part by making use of catalog information about the sensor network diameter, sampling rates, etc.

An envisioned usage is user walking up to a terminal and typing a request such as “Guide me to the nearest laboratory/room which is open, and has a non-occupied machine with a spreadsheet program”. Additionally the user might specify constraints such as noise or temperature levels, or maximum distance to the desired room. As the request is processed, the system tracks the movement of the person around the area and displays appropriate guiding information on screens near the person. We envision several other example uses:

- A computer administrator asking “How fast can I run my jobs, given 5 machines and a given power/cooling budget?”. The query will execute on each machine under load and measure throughput and temperature.
- A computer administrator specifying a constraint “Gracefully shut down machines near air conditioning outage or fire alarm”. The query will assess environmental conditions in the area of each cluster of machines, check the fire alarm status, and potentially issue commands to shut down or hibernate machines in danger.

<sup>1</sup>e.g., our sensor engine currently is limited to one join per active query.

Table 1: Division of labor in SmartCIS project

Student	Component
Mengmeng Liu	federated optimizer, stream optimizer and stream engine
Svilen Mihaylov	sensor optimizer, and sensor engine
Zhuowei Bao	stream SQL parser
Marie Jacob	catalog, and mesh networks
Rahul Iyer	sensor devices
Easwaran Subbaraman	GUI

- A robotics student asking “What is the location of the following mobile robot in the test are?”. The query will take advantage of on-board positioning system or nearby ceiling-mounted sensors and will potentially involve a complex interpolation function.
- A forgetful professor’s calendar: “Trigger a reminder and display it wherever I walk: my calendar and appointment schedule and my position in the building”.

## 4 SmartCIS Demonstration

The demonstration will include a combination of real, live components in Penn’s buildings (primarily the Moore School building, which holds our computing labs) plus remote laptops on-site at the conference. Refer to Figure 2 for the basic deployment and for a screenshot of our graphical interface.

The physical configuration of the SmartCIS devices and monitoring software at Penn are as described in Section 2. We will also maintain a “trace” of data from the various devices, as a fallback in the event that connectivity from the conference site fails: we can still execute Stream SQL queries over these traces.

On-site we will bring two laptops — each to display data and pose queries from the perspective of a different location within the building. We will also bring several motes, each of which will be logically mapped to an RFID sensor in the hallways of the building (i.e., its data will be used in lieu of the data from the actual device). When a visitor comes to see the demo, we will hand out a mote device performing as an active RFID beacon (transmitting a signal with low power) and have the visitor approach one of the local motes, simulating moving in the building. The visitor will then request a set of desired features for a free machine (e.g., Fedora, Word, etc.). The SmartCIS application will plot on the GUI a path in the building to an actual free machine in one of the laboratories.

Additionally, a visitor may click on different graphical elements of the GUI to see real-time information for the conditions in the demo area, e.g., machine state or laboratory status. Finally, we will include real-time information about the actual computations being performed: the query plan and its partitioning across sub-systems and devices; messages being exchanged among motes; wireless signal strength; remaining battery life; etc.

In summary, we believe that this demonstration will show the effectiveness of our ASPEN substrate and its research components in a real application, while simultaneously highlighting the potential of intelligent buildings even in the University setting.

## 5 Working Progress

Our SmartCIS project team includes six students: Mengmeng Liu, Svilen Mihaylov, Zhuowei Bao, Marie Jacob, Rahul Iyer, and Easwaran Subbaraman. Each one is particularly in charge of one of the components in this project. Division of labor is shown in Table 1.

So far, we have fully implemented and tested the following components:

**Parser** We have utilized Eclipse DataTools open-source codes to parse standard SQL Queries. It can parse a single "SELECT-FROM-WHERE-GROUPBY-HAVING" query into abstract syntax trees.

**Catalog** We have created a global catalog based on BerkeleyDB for our federated optimizer, stream engine and sensor engine to store and fetch general and statistical information about relations and streams, such as schemas, histograms, cardinality, selectivity, min/max values, etc.

**Federated optimizer** We have built a federated optimizer that takes the abstract syntax tree from our parser, and partitions it into segments which can be then sent to either the stream subsystem or sensor subsystem. Here we are able to validate a query segment on whether it can be executed by a subsystem or not. Based on its validity, we then use heuristics to determine where to execute the query. For instance, if the sensor engine is capable of doing in-network joins on two sensor relations, our federated optimizer would fulfill its capacity when necessary.

**Stream engine** A distributed stream engine has been developed to its full potential. It can take input data as a stream of insertions or deletions tuples, horizontally partition the data based their hash values, and execute queries through a number of stateful or stateless distributed operators, namely selection, projection, join, aggregate, groupby, fixpoint, union, scan, and ship.

**Sensor optimizer** We have built our cost-based sensor optimizer based on the goal of optimizing number of messages being transmitted in the sensor networks.

**Sensor engine** Our sensor engine is being built to support SPJ sensor queries, among which joins are done in-network via our proposed algorithms and techniques.

**Sensor devices** We have deployed our Crossbow Imote2 and IRIS motes and enable NesC capabilities on these motes. Users can write NesC applications and execute them on the connected sensor networks.

**Graphical interface** We have built a schematic building interface and a query interface. It shows the map of the building, with a certain number of laboratories, each one containing several machines with built-in softwares. Our graphical interface shows the position of motes in the building, a person's location, the current status of machines(occupied or not), room temperatures and so on. It allows users to specify their preferences through a user-friendly query interface.

The following components are being under development now:

**StreamSQL parser** We are working on enhancing the standard SQL parser to parse a subset of streamSQL queries, typically standard SQL plus some window specifications.

**Mesh networks** As another option of routing data in distributed stream engine, other than Pastry DHT, we are working on routing based on mesh networks given its potential wireless communication applications.

**Stream wrappers** We are also working on stream engine wrappers. For instance, wrappers for monitoring system CPU usage, for monitoring PDU power usage, for monitoring network usage, and for sensor data from the sensor base station, etc.

**TinyDB deployment on motes** Given that there are several sensor boards that have accelerometers, light, pressure, temperature and humidity sensors, there is only one which has GPS. But the data can be read in mote networks. We're currently trying to get TinyDb to run on those motes.

**More GUI functionalities** We are also working on actually displaying detailed machine information on the schematic building interface.

There are still a few tasks we have to accomplish in the future:

**Enhancing federated optimizer** We need to enhance our federated optimizer to be more flexible and extensible. Ideally it should be able to determine how to partition the global query and where to execute the sub-plan from its global cost-based metric based on global knowledge of subsystems through wrappers.

**Translating user preferences to SQL queries** Currently we don't have the capability to transform user preferences (from menu bars etc) to standard SQL queries. We need to build this component to connect to our current parser.

**Stream optimizer** We still need to enhance our stream optimizer to support statistics on streams, for instance, maintaining sketches on streams, and be able to compute joins between histograms, joins between histograms and sketches, and so on.

**Displaying real data** When all the components are finished, we can connect our executed results from the whole engine to display on our graphical interface, such as shortest routes to the specific machine, showing detailed info of a selected machine, etc.

We look forward to fulfilling all the capabilities and functionalities listed here in the future. We'll continue to work on these unfinished components and hope to finish our proposed demo by May 2009.

## References

- [1] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2), 2006.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD*, 2006.
- [6] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *VLDB*, 1997.
- [7] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, 2003.
- [8] M. Lenzerini. Tutorial - data integration: A theoretical perspective. In *PODS*, 2002.
- [9] M. Liu, W. Zhao, N. Taylor, Z. Ives, and B. T. Loo. Maintaining recursive stream views with provenance. In *ICDE*, 2009.
- [10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.
- [11] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. A substrate for in-network sensor data integration. In *DMSN*, August 2008.
- [12] 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.
- [13] J. Taft. The intelligent power grid. *Innovating for Transformation: The Energy and Utilities Project*, 6:74–76, 2006. Available from [www.utilitiesproject.com](http://www.utilitiesproject.com).