Design and Implementation of a Middleware for Sentient Spaces

Bijit Hore, Hojjat Jafarpour, Ramesh Jain, Shengyue Ji, Daniel Massaguer
Sharad Mehrotra, Nalini Venkatasubramanian,Utz Westermann

School of Information and Computer Science
University of California
Irvine, CA 92697
Email: {bhore, hjafarpo, jain, shengyuj, dmassagu, sharad, nalini}@ics.uci.edu, westermann@acm.org

***Abstract*— Surveillance is an important task for guaranteeing the security of individuals. Being able to intelligently monitor the activity in given spaces is essential to achieve such surveillance. Sentient spaces based on a large set of sensors provide the potential for such intelligent monitoring. However, heavily instrumenting a space with sensors it is not enough to build a sentient space. One needs a software architecture that allows programming all these sensors in a transparent and efficient manner. In this paper, we present SATware, a Stream Acquisition and Transformation middleware we are developing to analyze, query, and transform multimodal sensor data streams to facilitate flexible development of sentient environments. SATware provides a powerful application development environment in which users (i.e., application builders) can focus on the specifics of the application without having to deal with the technical peculiarities of accessing a large number of diverse sensors via different protocols.**

I. Introduction

Advances in computing, communication and sensing technologies has made it possible to create large scale physical spaces with diverse embedded sensors, ubiquitous connectivity, and computation resources that together can enable pervasive functionality and applications. Today, such applications are built in a somewhat ad-hoc manner directly on top of multimodal sensor streams and application writers are exposed to the technical challenges of accessing a large number of different sensor types via different protocols over a network. We envision a different type of pervasive computing infrastructure which we refer to as sentient spaces that provides powerful and flexible abstractions for building pervasive applications. In a sentient space, observers possess capabilities to perceive and analyze situation based on available multimodal data from disparate sources. Sentient spaces provide a much higher level of semantic abstraction compared to sensor middleware prevalent today. The key concept underlying sentient spaces is events. Events are essentially important occurrences (or state changes in a pervasive space) that are semantically meaningful for the application. For instance, entry/exit of people from a building might be a meaningful event in the context of a surveillance application. Instead of viewing data as sensor streams, sentient spaces provide the abstraction of the (semantically meaningful) event streams which form the building block for pervasive applications. Decoupling (and abstracting) events from the underlying sensors provides numerous advantages. First and foremost, it provides a level of separation that enables application writers to focus on the application logic instead of having to directly manipulate sensor streams. Another advantage is that it provides a framework for the underlying middleware and runtime to optimize application execution based on variety of application and resource constraints. For instance, an event (such as location of a person of interest in an instrumented space) might be detected through tracking built using video cameras. It could also be detected using coarse level triangulation using WiFi access points. Which specific sensing technique is utilized can be decided by the run-time based on available resources and the applications accuracy needs.

In this paper, we describe the architecture of SATware, a multimodal sensor data stream querying, analysis, and transformation middleware that aims at realizing a sentient system. SATware provides applications with a semantically richer level of abstraction of the physical world compared to raw sensor streams, providing a flexible and powerful application development environment. It supports mechanisms for application builders to specify events of interest to the application, mechanisms to map such events to basic media events detectable directly over sensor streams, a powerful language to compose event streams, and a run-time for detection and transformation of events. SATware is being developed in the context of the Responsphere infrastructure at the UC Irvine campus which is a unique publicly accessible testbed for interdisciplinary research in situation monitoring and awareness in the context of emergency response applications [7]. Currently, Responsphere includes more than 200 sensors of 10 different types deployed over the geographical space that covers about half of the UCI campus. The sensors range from network pan-tilt-zoom fixed video cameras, microphones, wireless motes with accelerometers, temperature sensors, and motion sensors to RFID tag readers and networked people counters. It also includes mobile, WiFi-connected sensors such as an autonomous vehicle

and a wearable EvacPack [1] carrying sensing devices like GPS devices, gas chromatographs, cameras, and microphones.

Responsphere is currently used to instrument and observe activities such as emergency drills that are conducted in the UC Irvine Campus. The data collected by sensors (e.g., video, audio, etc.) is used by sociologists to observe and analyze human behavior as well as ascertain the effectiveness of new technologies, protocols and strategies in crisis situations deployed within emergency drills. Furthermore, continuous data feeds from Responsphere sensors are used to build pervasive applications such as surveillance and infrastructure monitoring. Although Responsphere provides the main motivation of our work, it must be noted that SATware is a generic framework suiting any application scenario in which highly diverse data streams need to be processed.

The paper is organized as follows: using our needs in Responsphere as the main source of motivation, we highlight elementary notions and issues that multimodal sensor data stream processing middleware needs to address in Section II. In Section III, we briefly analyze related work with regard to which extent it addresses (or does not address) the raised issues. In Section IV, we describe the basic architectural building blocks of SATware and the underlying processing model. Section V, concludes this chapter with a summary and an outlook to ongoing and future developments of SATware.

II. Multimodal Sensor Data Processing: Notions and Issues

The processing of the sensor data produced by the large sensing infrastructure of Responsphere is a challenging task. The heterogeneity of the infrastructure and its dimensions provide a very powerful sensing infrastructure that enables a wide variety of applications. However, that same heterogeneity and dimensions undermines the ability to use it. Responsphere produces a set of heterogeneous data streams that range from raw video data to temperature readings. The sensor hardware, software, and their access protocols are also heterogeneous; and so are their mobility capabilities, power constraints, and processing, storing, and networking capabilities. This heterogeneity together with the wide range of applications that can be implemented in such an infrastructure and the fact that Responsphere encompasses a large number of sensors installed throughout the UC Irvine campus poses the following challenges.

Multimodal streams. The sensors in Responsphere produce a large diversity of data streams that need to be processed in a unified manner. Existing approaches to data stream processing focus mostly on a specific kind of stream: e.g., scalar streams as produced by motes (e.g., [1]) or event streams as produced by RFID tag readers (e.g., [6]). Only recently, the processing of raw media streams as produced by cameras and microphones have been considered as well (e.g., [5] [8] [9]).

[1]EvacPack is composed of a backpack with a small computer box with wifi connectivity, a pair of visualization goggles, a wireless mouse, a wearable keyboard, and a set of sensors including gas sensors, webcam, microphone, compass.

Abstraction. In order to provide an abstraction to ease sensor data stream processing in Responsphere and to allow automatic optimization of concurrent processing tasks, declarative languages for analyzing, querying, and transforming data streams are essential. Not all data stream processing approaches to sensor stream processing provide such languages (e.g., [10] [12]).

Scalability. In Responsphere, a large number of sensors potentially producing streams with considerably high data rates requiring substantial computational power for processing (such as video streams) pose the difficult challenge of efficiently utilizing both limited available network bandwidth and limited computing resources. Many of the sensor data processing infrastructures proposed in the literature, however, are based on centralized architectures thwarting scalability to Responsphere dimensions [5] [15] [20].

Extensibility. With the large number and variety of different sensors, developers face numerous challenges such as discovering which sensor types are available, what sensor streams they produce, where sensors are located physically and which area they cover, under which addresses they can be reached on the network, what the network topology is, what computing nodes are available for stream processing, etc. A sensor network directory service offering access to this information is required.

Mobility. The availability of mobile sensors in Responsphere constitutes a further challenge. While many sensor data processing infrastructures are capable of temporally synchronizing different streams for multimodal data using different variants of time window-based joins (e.g., [10] [14] [13]), mobile sensors also require location-based means of stream synchronization. For instance, one may want to join the camera streams of two autonomous vehicles only when they are observing the same area.

Power-awareness. Responsphere encompasses sensors that are battery-powered and sensors that are constantly connected to the power grid. Battery-powered sensors can also be further divided into rechargeable sensors and not (easily) rechargeable sensors. This heterogeneity implies that different power optimization approaches have to be considered when accessing each device and balancing the computation.

Privacy. As Responsphere covers a public studying and work environment, privacy of observed individuals is of high importance. To avoid misuse and unauthorized access to information about individuals, it must be possible to enforce privacy policies already at the level of the sensor data processing infrastructure. However, to our knowledge, privacy issues have not been playing a significant role so far in sensor data stream processing research.

III. Related Work

There has been extensive research in different areas of data management in sensor networks and pervasive environment that resulted in implementation of many systems. However, each of these systems has focused on a subset of the requirements described in Section II. In this section we provide a

brief overview of some of these systems.

Data Stream Processing. There has been a considerable work on data stream processing systems including Aurora [1], TelegraphCQ [11], and STREAM system [3]. These systems provide many features such as data model, continuous query semantic, query language, and scalable and adaptive continuous query processing. However, most of these systems have concentrated on data streams and do not consider multimedia streams.

Sensor Networks. Sensor networks generate high volume of data that need to be analyzed. Much work has been done on different aspects of data acquisition and analysis in sensor networks which expands a broad spectrum from issues such as energy saving to query accuracy [1]. However, most of the work in this field has focused on single type of sensors such as motes [21].

Multimedia Stream Processing. Recently there have been systems for processing multimedia streams including IBM Smart Surveillance System (S3) [15], MedSMan [5], IrisNet [8], and Smart Camera Network [9]. These systems provide features such as media capture, automatic feature extraction, declaration and query language, temporal stream operators, and querying algorithms. In this section we describe IBM S3 system and MedSMan in detail. The reader is referred to the cited references for a detailed description of the other systems.

The IBM Smart Surveillance System is a middleware that provides video based behavior analysis. It has two main components, Smart Surveillance Engine (SSE) and Middleware for Large Scale Surveillance (MILS). The main functionality of the SSE is to provide software based surveillance event detection. It uses video analysis techniques such as object detection, object tracking and object classification. The MILS middleware provides data management services for S3. It converts surveillance video into a relational database table. It also provides querying capabilities on the surveillance video. S3 also provides a privacy preserving surveillance feature that can hide personal information to different degrees. Despite all the rich functionality that it provides, the S3 system is based on a centralized database that may result in scalability issues in systems with large number of video sources. Also it only considers one stream type (video).

MedSMan is a live multimedia management system. It consists of three main components: Stream Model, Environment Model, and Event Model. The stream model in MedSMan consists of two basic parts: media streams and feature streams. A media stream consists of tuples with sequence number, a time interval and media content corresponding to that interval. A feature stream consists of a sequence number, a timestamp, and a feature value that has occurred on that time. MedSMan defines stream operators that can convert media streams to feature streams and vice versa. In MedSMan an event has a start and end time. MedSMan allows a hierarchical event structure where an event can consist of a set of events.

Based on its three main components, MedSMan defines a query language, which is an extension of CQL [10], to handle new concepts such as feature streams. Using this language, a user can define different queries on top of video streams and MedSMan evaluates them in real-time. In addition to video streams MedSMan supports audio streams; however, it does not provide an architecture that can be be extended to accommodate systems with large number of multimedia sensors.

IV. SATware Architecture

This section describes the basic building blocks of SATware. These building blocks, shown in Fig. 1, are organized as a stack of layers where each layer provides an extra level of abstraction of the sensing infrastructure. User applications write queries regarding the pervasive space being sensed (e.g., using a high level language such as CQL). These queries are translated into a graph of operators where each operator performs a function on a certain stream of data and produces another stream of data. Translating queries into operator graphs provides general applicability to the different stream data types we have, expressiveness in that other stream processing algebras can be mapped to this processing model, and natural mapping to a distributed execution. The main SATware layers are SATLite, SATDeployer, and SATRuntime.

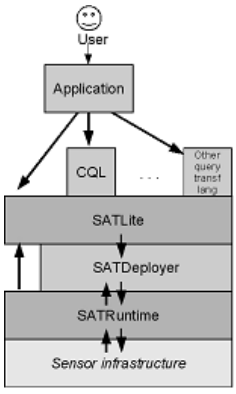

Fig. 1. SATware architecture.

SATLite provides a language and a language processor for describing graphs of operators. After the operator topology has been expressed in SATLite, each operator is assigned a machine where the operator will be executed. The mapping of operator graphs to machines and the deployment of such operators and establishment of their connections is done by the SATDeployer. SATDeployer uses the methods provided by the SATRuntime layer in order to deploy operators in the network. The SATRuntime layer is distributed along machines in a network (including sensors) and provides a runtime environment where operators can be injected and executed. Through the infrastructure directory, SATRuntime also provides an image of the available sensors, operators, and resources. This information is used by SATDeployer to optimize the operator deployment. The following subsections describe in more detail SATware's stream processing model, SATRuntime, SATLite, and SATDeployer. The last subsection introduces the concept of privacy in such a pervasive space and its implications for SATware's architecture.

A. Sentient building

Before presenting the details of each building block, let us first look into some details of our instrumentation and some sample applications. This will establish some context and aid in illustrating each of SATware's building blocks concepts.

As part of the instrumentation of the Responsphere pervasive space [7] we are instrumenting several buildings with sensors. In one of such buildings, the building where our offices and labs are located, we already have about 60 cameras (15 per floor) with microphones, an RFID reader with several antennas, several MicaZ wireless sensor nodes (with temperature, acceleration, light sensors, and so on), three mobile platforms with cameras and other sensors on them, and people counters on the doors of the building.

In such a heavily instrumented environment, we have implemented a building surveillance application to record the data being sensed: the SATRecorder. SATRecorder allows a user to visualize where the sensors are located, learn their current readings, and schedule recordings. It provides a GIS-like interface where the user can perform spatial queries with a graphical interface. Currently, SATRecorder allows a user to record video feeds based on a time-based schedule; in the future it will allow recording based on automatic event detection.

As part of the building instrumentation, we have also instrumented our kitchen space. Being smaller in size and close to our offices, our kitchen space provides us with a controllable and manageable pervasive space. It is in the context of this coffee room, where our proof-of-concept applications are being developed.

The Coffee Room Scenario

Throughout the paper, we use a sample application to illustrate different concepts and functionalities of SATware. In this application, a kitchen in a shared office space is instrumented with sensors which are used to monitor the resulting "smart kitchen", including the status of various appliances, in order to implement different policies of using a shared facility. In particular, video sensors and RFID readers monitor a coffee machine to determine policy violators. Examples of such policies include: "warn a person who leaves the coffee pot empty with the burner on more than three times", "charge people for number of cups of coffee they drink", "determine people who leave the kitchen area dirty", and so on. Fig. 2 depicts a snapshot from one of the cameras. Such events are used to realize the policies of our shared kitchen facility.

Note that the coffee room scenario can be extrapolated to monitoring other spaces such as buildings and airports. In these cases, the events become "leaving an object unattended", "exchanging bags", and other suspicious activity. However, given its simplicity and controllability, we chose to use the coffee room scenario for describing the different concepts in SATware. In addition, our instrumented coffee room, since it is easy to control and manage, provides us with a space to test SATware.

Fig. 2. A snapshot of coffee cam in smart kitchen application.

B. Stream Processing model

This subsection presents the multimodal stream processing model that enables processing of a broad diversity of sensor data streams in a scalable and flexible manner. First we present the stream data model adopted in SATware. We then present our data stream processing model that provides both flexibility and scalability.

The large number of sensors producing streams of data and the sharing of the same hardware infrastructure by several applications makes it indispensable to optimize the use of the infrastructure resources. The placement of each operator plays an important role in scalability of the system in terms of bandwidth, computing, storage, and energy. For instance, consider an operator that given an image of a coffee pot outputs the level of coffee in the pot and an operator that informs the user through a GUI about the amount of coffee in the coffee pot. Placing the image processing operator close to the video camera (i.e., in the same subnet or even in the camera itself) and the GUI at the user's PC (in a different network) is more bandwidth efficient than placing both at the user's PC. Thus, being able to distribute the operators along the network enables scalability.

In the coffee room scenario, we compose a query to determine if a user leaves the coffee pot empty with the burner on more than three times as the operator graph in Fig 3. In this example, we have i) an operator (O1) that gets video frames from the video camera, ii) an operator (O2) that gets id tags from the RFID reader, iii) three operators (O3, O4, and O5) that detect the status of the burner, the coffee level, and the coffee pot position, iv) three filters (O6, O7, and O8) that detect the events burner switched on, coffee pot level changed to empty, and coffee pot placed on the burner, v) an operator (O9) that detects the burning event if the coffee pot is on the burner, empty, and the burner is on, vi) an operator (O10) that given a person id and the burning event detects if a person has let the coffee burn more than three times, and vii) an operator (O11) that provides a GUI for the user.

1) Data model: In SATware, data streams go through a variety of transformations. To be able to define SATware's processing model, we first need to define the data model for the streams. A stream is an infinite discrete flow of packets and we define it as an infinite list of packets: *Stream = list(packet)*. A packet is a tuple of the form *packet = (t, c)*, where *t* is a timestamp and c is the packet content. The timestamp indicates the time that the packet content is related to; the packet content

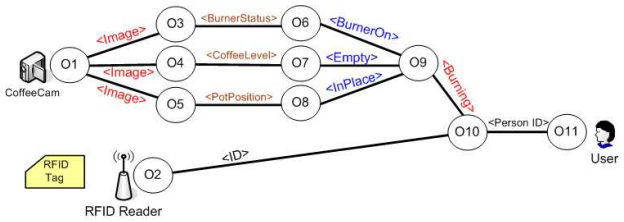

Fig. 3. A subgraph for detecting policy violation.

is the data the packet carries. The content can be either data of a simple type (i.e. integer, float, character, byte), a list or tuple of a simple type, or an event.

Events are more expressive than raw data and are either an interpretation of those or of other events. Given the inherent inaccuracy on hardware (sensing circuits, clock synchronization, etc.) and on software (sampling rate, approximations, stochastic event detections, etc.) events always carry with themselves a confidence level. Formally, an event is a tuple with the following form: *event = (eventID, confidence, DT).*

Apart from eventID that identifies an event and confidence level, events can also carry some extra data (*DT*) such as references to a stream where an event was originated from or quantifiers and qualifiers for the event.

2) Processing model: Streams are modified by operators. The set of operators is defined as a subset of functions such that every function has arguments of type DT, inputs of type Stream, and an output of type Stream: $OP = \{f_{DTx...xDT} \mid f_{DTx...xDT} : Streamx...xStream \rightarrow Stream\}$.

Virtual sensors. Operator subgraphs can be encapsulated into a single operator: a virtual sensor. This encapsulation simplifies the design of applications, and produces quicker and bug-free applications. Applications are easier to design since the complexity of the topologies decreases. Applications can be modularized and tested independently. Also reusability is increased since i) operator topologies (and not only operators) can be now reused, and ii) virtual sensors can be replaced without changing the rest of the application. An example of a virtual sensor is: WhoLeftCoffeeBurning = O10(CoffeeBurning, PersonID).

Using virtual sensor concept, SATware provides a dedicated notion of views for sensor data streams. The concept of virtual sensors is analogous to the data hiding principle in programming languages.

Operator graph. An application query can be viewed as a directed graph G such that G is a weakly connected graph formed by a set of vertexes V and a set of arcs A. We usually classify the nodes into source nodes, intermediate nodes, and destination nodes. Source nodes get data from sensors-they transform reality (or even a simulated or a predicted reality) into a stream of digitalized sensed values. These usually run at the sensor itself or on a PC in the same subnet. Intermediate nodes transform data streams. Among the intermediate nodes, we identify two special cases: transformers and synchronizers. Transformers are nodes that have only one incoming arc and synchronizers are nodes with more than one incoming arc. Synchronizers synchronize/join streams depending on values such as time or location. Destination nodes are not connected to other SATware nodes but produce output intelligible for a human user or other applications (e.g., it outputs into an XML file or a Database).

Physical deployment. Once an application query has been designed as a graph of operators, it has to be instantiated in the SATware system. Instantiating an application query means to determine which machine executes which operator. Cost functions are also defined for every operation. These functions express the cost of the operation in such terms as bandwidth, memory, CPU, and so on. Based on these assumptions the mapping function decides the optimal (or good enough) way to deploy operators given the cost of each operator and the current state of SATware in terms of deployed operators and availability of sensors and resources.

C. SATRuntime: The Stream Processing Runtime

SATware's runtime (SATRuntime) is a reflective distributed runtime which meets the above requirements. It has a central directory which contains the characteristics and state of each of SATware's resources (sensors, machines, and network), and a repository of operators and virtual sensors. Each operator is implemented as a mobile agent that can migrate to any of SATware's nodes. Mobile agents are autonomous software entities with the capability of dynamically changing their execution environments in a network-aware fashion [22, 23]. This adaptability allows operators to be deployed and dynamically redeployed as needed, to work in isolation when the network is temporarily down, and to migrate operators closer to where data is generated to optimize network resources. Moreover, the fact that agents can be thought as reactive autonomous entities that know how to perform certain actions (instead of just objects that encapsulate data) simplifies the application design.

Fig. 4 depicts the system architecture. The system nodes are of three types: a) sensor nodes, b) processing nodes, and c) the directory server. Sensor nodes correspond to the heterogeneous set of sensors in the pervasive infrastructure (e.g., Responsphere). These sensors range from RFID readers to video cameras and, thus, the programming platform in each of them is different. Due to this heterogeneity, SATware supports two ways of obtaining the sensed data. The first way is to create an agent gateway that connects to the sensor. For example, consider a network camera that runs a given operating system with a web server. A SATware agent can be programmed such that it opens an HTTP connection with the camera web-server and requests the video being captured. The second way to access the cameras is to extend the SATRuntime in the sensor node. For example, a compatible runtime (e.g., written in C or Java) is installed in the camera and a SATware agent is injected. The first allows for faster prototyping whether the second one brings the flexibility, adaptability, and robustness of mobile agents all the way to the sensor.

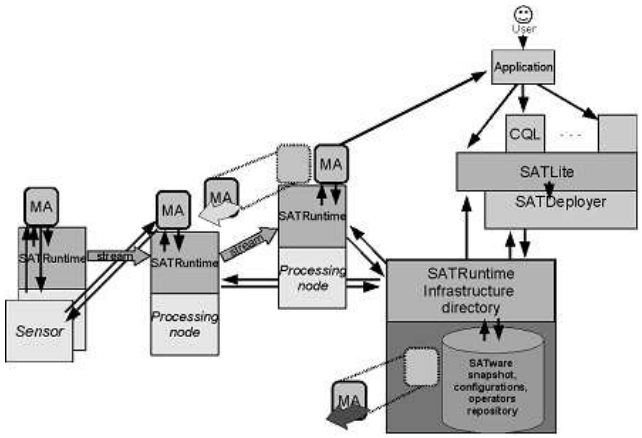

Fig. 4. SATware distributed architecture.

The second type of nodes in SATware are the processing nodes. SATRuntime is installed on each one of the processing nodes allowing agents to execute in each of them. SATRuntime provides mobility support and message passing to SATware agents. When deploying a graph of operators SATRuntime nodes (not the agents) are explicitly connected according to the topology. This way, agents are programmed in a topology-agnostic manner. Agents receive streams of data and output a stream of data, without needing to know the origin or destination of such streams. This simplifies the agent design and implementation, increases re-usability, and reduces implementation time, errors, and agent size. The third component of SATware is the directory server. Mainly, the directory server is an interface to the central directory. The directory server stores a repository of operators and the state of SATware. The repository of operators contains the mobile agent code that, when an agent moves into a SATRuntime node, is downloaded into the runtime if it had not been downloaded there before. The state of SATware contained in the directory server includes which sensors are available and how to access them, which processing nodes are available and how much resources they are offering, the network topology and its state, and the current agent deployment. Interfaced via web services, the directory server provides services such as insert, query, and update types (e.g., stream types, sensor types, machine types, router types, operator types) and instances (e.g., sensors, machines, routers, operators). It also provides web services for deploying operators and graphs of operators. SATRuntime and other components such as the query processor interact with the directory server so both application and processing nodes constraints (e.g., bandwidth, storage) can be met and optimized. In addition to assisting on deploying applications, the directory server provides a permanent storage area where SATware agents can save persistent data (e.g., the encrypted automatons described in Section IV-F).

D. SATLite Stream Query and Transformation Language

SATLite stream query and transformation language in SATware provides an interface for describing and instantiating operator graphs. Stream querying and transforming are realized by SATLite language together with SATware agents. There are two types of SATLite users: a user that types commands in SATLite language to deploy, move or terminate agent operators on the network, and an administrative agent that generates SATLite statements to control the running of other agents on different machines. The former usage facilitates the implementation of a system dashboard; the latter could support query plan generator for higher level continuous query languages such as CQL [10] and TelegraphCQ [11].

Consider the smart kitchen scenario. Our purpose is to detect whether someone left the coffee pot empty and burning on the coffee machine more than three times. Let us assume that functions that provide certain transformation and synchronization functionality already exist. For example, function *ReadCam* reads a video stream from a specified camera; function *DetectBurner* detects whether the burner of the coffee machine is switched on; function *And* does the logical and operation on multiple boolean inputs.

The first step is to create an operator instance from existing functions for each corresponding position on the graph. As shown in this example there may be multiple instances of a single function running on different locations and/or running with different inputs/outputs and parameters. So it is necessary to assign an identifier for each running instance of the function. After we have deployed all the agent instances, we connect them together using channels. A channel is a data path that has one source and one or more destinations. Channels serve as input and output variables for agent functions. Stream compatibility check in SATLite is similar to type checking in a programming language. Before such checking is issued, agent functions (not instances) need to be registered into the directory service with their input/output stream type. Stream declaration will be checked whenever SATLite receives a data flow setup statement. If compatibility is not satisfied, the statement will be rejected. Notice that by specifying the keyword *'any'*, the expected stream is compatible with any stream types.

E. SATDeployer: Operator Deployment in SATware

SATDeployer is the operator deployment component in SATware, responsible for deploying stream processing operators. SATDeployer translates a query plan (topology) described in the SATLite language into a deployment plan. Apart from the query plan, SATDeployer needs information about the current operator deployment, the available sensors, and the state of the network and machines available to SATware. Different objectives can be defined for operator placement in a distributed stream processing system. Minimizing communication cost, latency, or operator computation cost are some of the objectives that can be considered in operator deployment for executing queries.

The main step of query deployment in SATDeployer is to map the query plan graph to the nodes in the network. The result of this mapping is an operator deployment graph. Each node of the deployment graph corresponds to a machine in the

SATware network and depicts the operations that should be performed in that machine. The deployment graph is formed in such a way that the objective of the operator placement is satisfied. After forming the deployment graph, SATDeployer, using the API provided by SATRuntime, sends the operators to the corresponding nodes in the SATware network and connects them to each other according to the deployment graph. Then the stream processing to answer the query starts.

Consider a query to detect people who do not pay attention if the pot is burning. The query should be as follows:

SELECT PersonID from KitchenEvents where PotBurning = true;

A possible instantiation of this query would use streams from two sensors, an RFID reader and a camera. From the RFID reader we obtain the identity of who is in the kitchen. With the camera, we detect whether the coffee pot is on the burner, the burner is on, and the coffee pot is empty. When there is someone in the kitchen and the pot is left empty on the burner and the burner is on, we want to get the identity of the person who was in the kitchen. The query plan generated by SATLite for this query is depicted in Fig 3.

After receiving the query plan from SATLite, it is the responsibility of SATDeployer to decide which operator should be placed on which machine. Consider a network with three processing nodes. One possible deployment plan is to perform O3, .. , O9 on the Processing node 1, O2 on Processing node 2, and O10 on the Processing node 3. However, depending on the network situation and other existing operations SATDeployer may decide to place O3,.., O9 on the Processing node 3.

Much research has been conducted on the operator placement problem in recent years [16] [17] [18] [19]. Most of the existing work in operator placement concentrates on minimizing communication load including bandwidth consumption and latency and it ignores the processing load caused by performing operators in the nodes. However, in SATware applications there are operations, such as event extraction from video stream, that have a high processing load which cannot be ignored.

The other important feature that SATware considers is the reuse of the existing operators in the network. Many incoming queries might share some operators. In all of the existing operator placement systems the placement for each query is done independently, which may result in replicated execution of shared operators among queries. This results in higher processing load on the nodes. Since SATDeployer has the knowledge of the deployment graph of the existing queries, by reusing existing operators it can considerably reduce the processing load in SATware.

F. Privacy

In this section we illustrate how the SATware system can be configured to implement human-centric applications in a privacy-preserving manner. To illustrate, let us consider individuals in the pervasive space have roles assigned to them such that a set of policies apply to each role, e.g., students can have at most three free cups of coffee in a day, professors will need to pay for each cup. The goal of a surveillance application could be to detect events that violate such policies. In general, policy-violating events are *composite events*, composed of one or more sequences of *basic events*. We can denote such a composite event by a finite-state automaton where the transitions between its states happen on some basic event. An application will have to maintain partial information about the "state" (automatons) of the pervasive environment in order to detect the occurrence of policy-violating composite events. Therefore, the question that arises is the following: "Is it possible to design an event-detection system in a manner that reveals an individual's identity (if at all) only when he violates a policy applicable to him and not at any time before that".

Fig. 5 shows an automaton for detecting the event when an individual has had more than three cups of coffee. S_0 and S_F denote the start and the end states of the automaton respectively. The basic event e1 = $<$ u, coffee-room, coffee-cup, exit $>$ denotes the act of leaving the coffee-room with a cup of coffee. The self-loops labeled $\neg$e1 denote all other basic events and act as filtering edges.

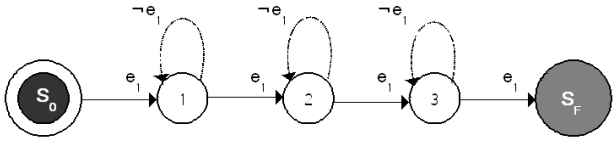

Fig. 5. Automaton for detecting that an individual has had more than three cups of coffee.

We assume the adversary to be an inquisitive but non-interfering (passive) entity that can observe all stored data and intercept all communication between two components of the system. Such an adversary is also assumed to be knowledgeable of the policies and the corresponding automatons that implement these policies. In real life, such an adversary could be an insider, like a system administrator.

To keep things simple, we will restrict the notion of privacy simply to that of anonymity and the application semantics to that of implementing deterministic finite-state automatons (representing composite events). We say the system ensures k-anonymity of individuals if no observable pattern (of automaton access/manipulation plus stored data/meta-data) leads to identifying an individual to within a set of at most k individuals. The problem now is that of designing a system that can store, retrieve, and advance automatons in a k-anonymous manner, i.e., where the adversary can at best attribute any event *e* in the stream to a set of *k* or more individuals even after observing a very large number of events after e.

A generalization/clustering based scheme to anonymize event streams for a surveillance application are discussed in detail in [24] which is easily applicable in the SATware system as well. But, unlike in that case, the trusted nodes in the SATware system will also require implementing operators that can scrub data streams of identifying information in such a manner that other (untrusted) nodes are able to perform their tasks on the resulting stream. For example, a video stream

containing an individual as well as the coffee pot might be processed at a trusted node first which scrubs (hides) the human figure from the video frames (Fig 6) before passing on the stream to another node that is responsible for detecting whether the coffee pot is empty or full.

Fig. 6. Human identity anonymized on a video frame.

V. Conclusions and Future Work

Sentient spaces have the potential to allow intelligent privacy-preserving surveillance. Such spaces are permeated with multimodal sensors that capture the environment and the activities within. In contrast to most sensor middleware prevalent today, sentient spaces' users are provided with streams of events rather than streams of data. Events provide a higher abstraction than raw data and are semantically meaningful. The higher level of abstraction provided by sentient spaces empowers users with the capabilities to monitor entities (such as spaces and individuals) without needing to know (and deal with) the details of an underlying heterogeneous sensor network.

This paper presents and discusses each of the challenges involved in building a sentient space. Given that no current system addresses all of them, we then propose SATware: a middleware for sentient spaces. SATware's basic blocks include a distributed runtime system (SATRuntime) that permits injection, execution, and interconnection of data stream processing operators, a declarative language (SATLite) for composition of such operators, an operator deployment module (SATDeployer), and an infrastructure directory for storing the availability of resources.

Currently, SATware is being implemented in a pervasive space being build in our campus. There, we have instrumented a shared kitchen space with a set of sensors including two D-link wireless Internet cameras, two Linksys Internet IP webcams, and an RFID reader. Communication between different parts of system is done through IEEE 802.11 and 100Mbps Ethernet LANs. Our first applications and proof-of-concept demos have been developed in that kitchen.

As part of our on-going work, we are building larger applications that involve the rest of the instrumented space. Future work involves the research and implementation of optimization algorithms for operator deployment, as well as supporting different network protocols between SATRuntimes to cope with different types of data. Last, privacy is an important issue that has to be taken into account when building pervasive spaces. Here we provide a technical solution for preserving privacy when detecting human-centric events. However, our solution only deal with a subset of all possible problems. Investigating further privacy preserving related issues is also part of our future research.

References

[1] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong. *TinyDB: an acquisitional query processing system for sensor networks.*, ACM Trans. Database Syst. 30(1): 122-173 (2005).

[2] U. Srivastava, K. Munagala and J. Widom. *Operator Placement for In-Network Stream Query Processing.*, ACM PODS 2005.

[3] D. Abadi, D. Carney, U. Cetintemel, and et al. *Aurora: A new model and architecture for data stream management.*, VLDB Journal, 12(2):120139,August 2003.

[4] Paolo Costa and Gian Pietro Picco and Silvana Rossetto. *Publish-Subscribe on Sensor Networks: A Semi-probabilistic Approach.*, Proceedings of the 2nd IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS05), Washington DC, USA, 2005.

[5] B. Liu, A. Gupta, R. Jain. *MedSMan: a Streaming Data Management System over Live Multimedia.*, In Proc. of the 13th ACM Intl. Conference on Multimedia (MULTIMEDIA 05), Singapore, Nov. 2005.

[6] Joseph M. Hellerstein, Wei Hong and Samuel R. Madden. *The sensor spectrum: technology, trends, and requirements.*, ACM SIGMOD Record ACM SIGMOD Record, Volume 32 , Issue 4 (December 2003)

[7] http://www.responsphere.org/

[8] http://www.intel-iris.net/research.html

[9] http://wsnl.stanford.edu/index.php

[10] Arvind Arasu, Shivnath Babu, Jennifer Widom. *The CQL continuous query language: semantic foundations and query execution.*, VLDB J. 15(2): 121-142 (2006)

[11] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Frederick Reiss and Mehul A. Shah. *TelegraphCQ: Continuous Dataflow Processing.*, SIGMOD 2003.

[12] A. Demers, J. Gehrke, M. Hong, et al. *Towards Expressive Publish/Subscribe Systems.*, In Proc. of the 10th Intl. Conference on Extending Database Technology (EDBT 2006), Munich, Germany, March 2006.

[13] A. Arasu, S. Babu, J. Widom. *An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations.*, Technical Report 2002-57, Stanford University, Stanford, California, 2002.

[14] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. *Models and issues in data stream systems.*, In Symposium on Principles of Database Systems, pages 116, Madison,Wisconsin, May 2002.

[15] http://www.research.ibm.com/peoplevision/index.html

[16] Z. Abrams and J. Liu. *Greedy is Good: On Service Tree Placement for In-Network Stream Processing.*, IEEE ICDCS 2006.

[17] X. Gu, P. Yu and K. Nahrstedt.*Optimal Component Composition for Scalable Stream Processing.*, IEEE ICDCS 2005.

[18] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. *Network-Aware Operator Placement for Stream-Processing Systems.*, IEEE ICDE 2006.

[19] L. Amini, N. Jain, A. Sehgal,J. Silber, O. Verscheure.*Adaptive Control of xtremescale Stream Processing Systems.*, IEEE ICDCS 2006.

[20] D. Carney, U. Centintemel, M. Cherniack, et al. *Monitoring Streams A New Class of Data Management Applications.*, In Proc. of the 28th Intl. Conference on Very Large Data Bases (VLDB 2002), Hong Kong, China, Aug. 2002.

[21] Crossbow Technology, Inc. MPR/MIB User's Manual, 2004.

[22] Karnik, N. M. and Tripathi, A. R. (1998), *Design Issues in Mobile-Agent Programming Systems.*, IEEE Concurrency, Vol. 6, No. 3, pp. 52-61, July-September.

[23] Fuggetta, A., Picco, G. P. and Vigna, G. *Understanding Code Mobility*, IEEE Transactions on Software Engineering, Vol. 24, No. 5, pp. 352-361, May 1998.

[24] "Privacy-Preserving Event Detection for Media Spaces" ICS-UCI technical-report, 2006.