Capturing High-Level Conditions, using a Publish/Subscribe Middleware, in Sensor Systems

Salman Taherian and Jean Bacon

University of Cambridge
Computer Laboratory,
JJ Thomson Avenue, Cambridge, CB3 0FD, UK
{st344,jmb}@cl.cam.ac.uk

Keywords: states, publish/subscribe, sensor systems

Abstract

In this paper we discuss State Maintenance Components (SMCs), which are used to capture high-level conditions in a State-based Publish/Subscribe (SPS) middleware. SMCs support temporal and spatial condition interrelationships, while using events received from a Publish/Subscribe (Pub/Sub) communication system. The SMC data structure is designed so as to enable SMC decompositions and distributions for effective load-balancing, localized processing, and information sharing. Our evaluation, using real sensor data in the context of a smart transportation system, demonstrates the expressiveness and the scalability of SMCs in the SPS middleware.

1 Introduction

With the advance of sensor technologies, and increased realisation of the benefits of studying environmental data, new sensor systems are emerging, that exhibit sheer scale, heterogeneous device types, and multi-application operational settings. Smart environments are an example of these systems, where environments are equipped with wired/wireless sensor devices of various types and platforms, and serve diverse and dynamic applications.

These systems demand middleware solutions that abstract the low-level data, and the infrastructural details, and capture high-level knowledge or conditions for their applications and users. Most important, however, is the dynamic topology that is realised in these systems. This is not only related to node failures, but also to network characteristics (e.g. node mobility), the multi-application setting (distributed and dynamic users, with changing interests), and the system's evolution (node joins and leaves).

Provisioning the highlighted information processing needs, in the view of the dynamic network topology, is a great challenge. In an effort to address this, we have decided to leverage from existing Pub/Sub systems[4], which have been extensively researched in the context of large-scale distributed systems. The aim is to build a suitable information processing service that cooperates with the Pub/Sub communication paradigm.

Capturing high-level conditions, using a Pub/Sub system, is non-trivial. Publishers' details, such as the quantity and granularity of their publications (e.g. rate of events), are hidden from the subscribers and variable throughout the system lifetime. Related efforts have led to the development of Composite Event (CE) frameworks, that are further discussed in the next section. We, however, have adopted the notion of *state* to capture lasting conditions in the system. Our efforts have led to the development of the SPS framework[17], which combines the expressiveness of states with the scalability of Pub/Sub systems.

The SPS middleware builds on stateless topic-based Pub/Sub protocols, which are simple in design and lightweight for operation on resource-constrained devices. More resourceful devices house State Maintenance Components (SMCs), that capture high-level conditions. These components maintain limited data structures that aid context-based data processing (for increased efficiency), and memory-based condition detection (for increased expresssiveness). In this paper, we focus on SMCs, and discuss their expressiveness and operation for capturing high-level conditions in sensor systems.

In section 2, we present an overview of related work. Section 3 outlines our middleware architecture; further details of which can be found in [17]. Capturing expressive conditions, using SMCs, is detailed in section 4. Section 5 evaluates the expressiveness and performance of the SPS middleware, in the context of a smart transportation system case-study. Finally, concluding remarks and future directions are outlined in section 6.

2 Related Work

SPS is not the first framework to use the notion of state for sensor systems. Others [6, 8, 1] have offered the expressiveness of states to sensor network applications. But they are mainly based on the principles of Finite State Machines (FSMs), and describe the internal state of a program in sensor networks. They are predominantly "state-oriented programming models", in which one or more user applications can be modelled and programmed over sensor devices. Other works use the notion of state to reflect knowledge about the real-world. Examples include [18] where lasting conditions are captured over correlated events, and [14] where high-level information is deduced from primitive state events. In [14], primitive state events are drawn to a centralized server, where expressive state predicates are

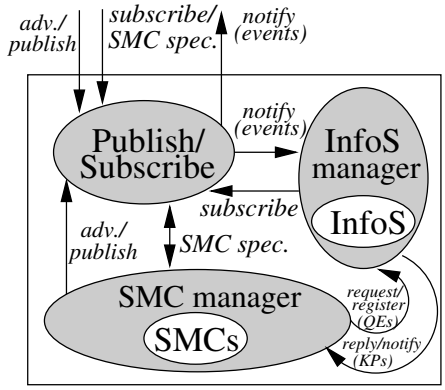

Figure 1: SPS architecture

evaluated. Our work uses a similar notion of state, but offers SMCs that are more expressive in describing conditions, and decomposable for distributed processing.

CE frameworks [13, 12, 2, 7] extract high-level information through patterns of event occurrences. These patterns are enveloped as individual composite events, which may subsequently serve as events to other composite events. These frameworks focus on temporal relationships between events. Event parameterization (constraints over event attribute values) hinders the sharing of CEs, and is often performed pre- or post-CE detection. Selection and discard of constituent events are governed by *per topic* event consumption policies. In contrast, our framework supports event parameterization as part of its condition detection process, and selects and discards events according to their contained knowledge (attribute values). SPS defines a unified model for supporting *per condition* temporal and spatial policies that govern the event selection, condition detections, and condition interrelationships. Furthermore, unlike some CE frameworks, SPS detectors (SMCs) can capture multiple concurrent conditions without the need to have pre-existing replicated detectors.

With a Database-oriented view on sensor networks[5], DB-related frameworks [10, 19, 9, 15] have been developed, that support application-level Structured Query Language (SQL) queries over resource-constrained sensor devices. SQL queries are parsed into procedural Relational Algebra Expressions in Database Management Systems (DBMSs). Although we use Relational Algebra to define our condition detection model, our framework is *state-based* (maintains states, provides context-based data processing, and facilitates memory-based condition detection) and is conceptually different from the work on DBMSs and data-stream processing, focusing on an open distributed environment. We describe how temporal and spatial condition interrelationships can be captured using Relational Algebra, and in this process, the efforts of the DB research community can be used for accurate optimization and implementation.

3 State-based Publish/Subscribe (SPS)

SPS[17] is a State-based Publish/Subscribe framework that supports its clients, comprising sensors, actuators, applications, controllers, etc., through a *uniform Pub/Sub interface*. An information producing client is called a *publisher*, and a consuming one is called a *subscriber*.

Table 1: SMC Structure

SMC	
Index	Annotation
N	SMC Name
Q	SMC Query Expressions (QEs)
P^n	SMC Entrance Predicate
A^n	ingress Event Attributes Computations
P^x	SMC Exit Predicate
A^x	egress Event Attributes Computations
s	SMC Status-bit
e	SMC's last Capture (Event Publication)

Figure 1 shows the internals of the SPS middleware: the *Publish/Subscribe*, the *Information Space (InfoS) manager*, and the *SMC manager* components. There is an explicit separation of tasks: the Pub/Sub component handles network-wide messaging, the InfoS component serves as an event repository, and the SMC manager captures high-level conditions. This separation helps to isolate the costs and predict run-time resource usage. Information flow, in SPS, is by means of event notifications, comprising a set of attribute/value pairs. Some attributes (topic, time, location, and status[1]) are generic, while others are topic-related. High-level information deduction, in SPS, is through SMCs, that have a structure outlined in table 1. Every SMC is an instantiated object that envelopes the condition definition in $(N, Q, P^n, A^n, P^x, A^x)$, and maintains a status-bit s and SMC event e that reflects the current status and last capture of the condition. The status-bit s and the last capture e can be examined through the $valid$ and $last$ keywords within the SMC predicates.

SMCs are housed at the SMC manager, and operate as publishers to the Pub/Sub component. Their names indicate the event topics under which their events are published. QEs highlight the relevant data that is obtained from the InfoS to examine the conditions. The entrance predicate captures the *start* of a condition, for which an ingress event (with related attributes) is generated, and the exit predicate captures the *ending* of a condition, denoted by an egress event. Context-based data processing is performed by oscillating between the two predicates, according to the current status of the condition, s. The next section details how SMCs are evaluated and conditions captured in SPS.

4 Capturing Conditions

In SPS, the notion of state is used to capture momentary or lasting conditions. We say that a condition has occurred if there exists some knowledge that contributes or hints about its occurrence. The detection process can be divided into three stages: *knowledge selection*, *knowledge examination*, and *knowledge encapsulation*. Throughout this process the following data structures are observed, that refine the InfoS data to some knowledge that suggests the condition's occurrence, and finally contribute to an SMC event.

Knowledge Point (KP) is an event-like attributed tuple, that reflects a piece of knowledge in the InfoS.

Satisfying Knowledge (SK) is a set of KP-combination

[1] this attribute may have the *atomic*, *ingress* or *egress* value.

tuples, each of which represents a combination of one or more KPs, whose values satisfy an SMC predicate.

Contributing Knowledge (CK) is a set of KP-combination tuples, that satisfy an SMC predicate and relate to a unique condition.

We explain the detection process with reference to an example. A high-level explosion event is defined as below.

Explosion Event. An explosion event is detected when sound and light events are detected, and high temperature readings are realised from the environment. The time of the sound and light events should be close (within $0.5s$), and high temperature readings should be detected continuously for the past $30s$. Multiple sound events, realised at different times, could relate to different explosion events.

4.1 Knowledge Selection

QEs extract distinct pieces of knowledge from the InfoS, to be examined individually or against other knowledge for condition occurrences or terminations. Every QE resolves into a set of KPs, when examined against the InfoS.

If we consider the InfoS as an MDX[11] *cube*, with dimensions matching the fixed event attributes, then each QE could be thought of as a selection box, that encloses some related knowledge in the cube, represented later by some KPs. The QEs are composed of attributed parameters that govern the selection of knowledge along every dimension of the cube. These parameters hold a one-to-one relationship with the fixed event attributes. Table 2 shows the set of permissible values for each attribute. D_P is the set of all event topics, D_T, D_L are the set of time and location values, and $D_S = \{atomic, ingress, egress\}$. The $<$ `range` $>$ parameters are described using a pair of values (r_l, r_h) that denote the lower and higher bound values. The default `range` is $(-\infty, +\infty)$. QE parameters follow one-another with a dot ('.') symbol in the representation. For example, the "$I(Temperature).L(-30^r, 0^r).N.I(atomic)$" is a QE, that holds the $I(Temperature)$ as its *topic* parameter, the $L(-30^r, 0^r)$ as its *time* parameter, the $N(-\infty, +\infty)$ as its *location* parameter, and the $I(atomic)$ as its *status* parameter.

If we consider the InfoS as a Database (DB) relation, then the parameters could be described using relational algebra. Prior to this, however, we must define the domain (set of permissible values and their meanings) for each attribute in the InfoS. For every fixed attribute, we define two domains: an *absolute domain* D and a *relative domain* D^r. The absolute domain reflects the set of static values that are storable in the tuples of InfoS. The relative domain, however, offers an alternative and dynamic view to the same set of static values in D. Queries that use relative values (e.g. $\sigma_{time==0^r}$ `InfoS`, where 0^r is a relative value) may return different relations, when examined by different queriers, at different times, or at different locations.

The relative domain D^r can be best explained by mapping each attribute domain D onto a unique domain of real numbers. Note that these domains are used as part of describing QEs that have corresponding SMCs, who also have corresponding hosts in the network.

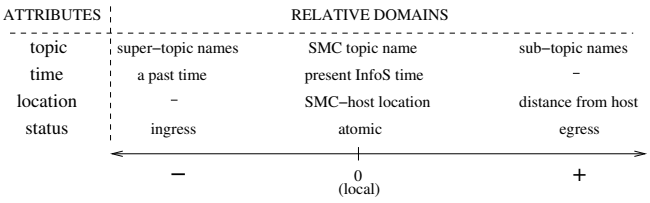

Figure 2: Relative domains

Topic The event topics from the absolute domain are mapped onto the domain of real numbers based on the corresponding SMC's name and super- and sub-topic names (e.g. *explosion* event topic is a sub-topic of *light*, *sound*, and *temperature* event topics).

Time The time values are mapped according to the local InfoS's processing timeline, that monotonically increases as knowledge (events) become stable[17].

Location The location values are mapped according to their Euclidean distances from the position of the corresponding SMC's host.

Status The status values are mapped statically.

Figure 2 illustrates this mapping over an axis of real numbers. Let $A_F = \{topic, time, location, status\}$ represent the set of fixed event attributes, and A_T represent the set of topic-related event attributes. We now define the QE parameters, using relational algebra, about a candidate attribute $a \in A_F$. Note that relative values are expressed like absolute values with an r superscript.

$I(v)$ $\bigcup_{(\{x|x\in A_F, x\neq a\}\gamma_{min(|a-v|),\{y|y\in A_T\}}\,\texttt{InfoS})}$, means that the knowledge closest to or at the v index value is selected (e.g. for the time attribute, $I(0^r)$ selects the most recent knowledge from the InfoS).

$A(\texttt{fn}, <\texttt{range}>)$ $\bigcup_{(\{x|x\in A_F, x\neq a\}\gamma_{\texttt{fn}(\{y|y\in A^T\}),a}\,Y)}$, where $Y = \sigma_{r_l\leq a\leq r_h}\,\texttt{InfoS}$. This parameter selects all knowledge, that falls within a given range, and applies an aggregation function, `fn`, independently of the other fixed attributes. Maximum (max), minimum (min), sum (sum), and average (avg) are supported.

$\{O, L, N, S\}(<\texttt{range}>)$ $\sigma_{r_l\leq \texttt{InfoS}.a\leq r_h}\,\texttt{InfoS}$, means that all knowledge instances, falling within the specified range, are selected from the InfoS.

If we label the resultant relation, from resolving a QE parameter, corresponding to an attribute $a \in A_F$, as $R_a(X)$ where X is the input relation, then the overall KPs for a QE is $K = R_{topic}\,(R_{status}\,(R_{location}\,(R_{time}(\texttt{InfoS}))))$. The earlier QE example retrieves the past $30s$ temperature readings, published by all the sensors across the space.

4.2 Knowledge Examination

SMCs are examined whenever their related knowledge is changed or updated. This change is not restricted to the reception of a new event, but may occur when the processing timeline is advanced, or the SMC's host changes position. The change triggers the InfoS to provide new KPs, which are then examined for condition occurrence or termination. The predicates define what knowledge contributes or hints about a condition's occurrence. If we label an SMC's QEs as $A, B, C, \ldots$, and their corresponding KPs as $K_A, K_B, K_C, \ldots$, then the SK S can be found as follows.

Table 2: Query Expression attributes and parameters

Attributes	Parameters	Default
topic	$single(I)(v \in D_P \cup D_P^r), multiple : \{one(O), all(L), any(N), separate(S)\}(<\texttt{range}>)$	–
time	$I(v \in D_T \cup D_T^r), aggregate(A)(\texttt{fn}, <\texttt{range}>), multiple : \{O, L, N, S\}(<\texttt{range}>)$	$I(0^r)$
location	$I(v \in D_L \cup D_L^r), A(\texttt{fn}, <\texttt{range}>), multiple : \{O, L, N, S\}(<\texttt{range}>)$	$N(-\infty, +\infty)$
status	$I(v \in D_S \cup D_S^r), multiple : \{O, L, N, S\}(<\texttt{range}>)$	$I(ingress)$

$S = \sigma_{p \in \{P^n, P^x\}}((\rho_{\{A.topic, A.time, A.location, \cdots\}} K_A) \times (\rho_{\{B.topic, B.time, B.location, \cdots\}} K_B) \times \cdots)$, where p is selected according to the current status of the condition.

If K_A and K_B represent the light and sound events, and the SMC predicate is $|A.time - B.time| \leq 0.5$, then S would contain all light and sound event-combinations whose occurrence times are within $0.5s$. This can be used as part of a search for detecting explosions.

The cartesian product of the relations $(K_A, K_B, K_C, \ldots)$, and the examination of all KP-combinations may be computationally expensive. In section 4.4, we discuss the decomposition of QEs, which allows this computation to be distributed across many networked devices.

4.3 Knowledge Encapsulation

Every tuple in S highlights a KP-combination that satisfies the predicate. The user specifies how these tuples should relate to one-another, and how they may be grouped to indicate one or more condition occurrences or terminations. Knowledge encapsulation is a process whereby the SK S is transformed into zero or more CKs. The number of resulting CKs determines the number of detected conditions.

The cardinality of S relates to the cardinality of the input KPs $(K_A, K_B, K_C, \ldots)$, such that if $|S| > 1$ then $\exists x \in \{A, B, \ldots\}$ where $|K_x| > 1$. For $|K_x| > 1$ to hold, the corresponding QE, x, must hold at least one $multiple$ parameter among its attributed values. We define sub-parameters for $multiple$, that define what relationship the resultant KP-combinations in S should hold for the condition to have occurred. Assuming that the $multiple$ parameter relates to an attribute $a \in A$, then the sub-parameters and their associated conditions may be defined as follows.

multiple:one (O) $|\pi_a S| = 1$. Asserts that only one unique a value, from K_x, should appear in S.

multiple:all (L) $|\pi_a S| = |\pi_a K_x|$. Asserts that all a values, from K_x, should appear in S.

multiple:any (N) $|\pi_a S| \geq 1 \equiv |S| \geq 1$, is a dummy parameter that asserts any one satisfied a value, from K_x, may be taken as a representative in a CK U.

multiple:separate (S) $|\pi_a S| \geq 1 \equiv |S| \geq 1$, is similar to $multiple : any$ but implies that every unique a value in S can signal a unique condition. A maximum number of $|\pi_a S|$ conditions can be captured per evaluation.

A CK U is a subset of the SK S, such that all conditions (related to the $multiple$ sub-parameters in x) are satisfied within U. These sub-parameters can be used to define temporal and spatial condition interrelationships in SPS. For example, continuous high temperature readings can be asserted using the $multiple : all$ parameter for the time attribute of a QE like C that acquires temperature readings from the InfoS. Similarly, distinct explosion events (related to different sound events) may be captured using the $multiple : separate$ parameter for the time attribute of a QE like B that extracts the sound events from the InfoS. The *Explosion* SMC structure is shown in table 3.

Table 3: Explosion SMC

SMC	
N	"Explosion"
Q	$A := I(Light).N.N.I(atomic);$ $B := I(Sound).S.N.I(atomic);$ $C := I(Temp).L(-30^r, 0^r).N.I(atomic);$
P^n	$\lvert A.time - B.time\rvert \leq 0.5 \,\&\&\, C.value \geq 40$
P^x	$true$

In order to determine all CKs (from an SK S), the SK S is divided into CKs according to the $multiple : separate$ parameter, $\{U | U = \sigma_{(a=i \in (\pi_a K_x))} S\}$. The $multiple : one$ and $multiple : all$ parameter assertions are then examined over each set member, and unsatisfied CKs are eliminated from the $\{U\}$. Subsequently if $|\{U\}| > 1$, then concurrent conditions are detected, and temporary SMCs are spawned to monitor each distinct condition individually. Every temporary spawned SMC is assigned a unique CK $u \in \{U\}$, and lasts until its assigned condition is terminated.

SMCs transform CKs into SMC events. The most recent knowledge in the CK and the SMC's attributes computations, A^n/A^x, are used to determine the topic-related attribute values. The generic attribute values are determined from the SMC's name, the most recent KP in CK, the location of the SMC's host, and the satisfied predicate. The condition capturing process is completed when the SMC event is published and the SMC status-bit is toggled.

4.4 Distributed Detection

SPS supports the decomposition of SMC to distribute the processing load across many network nodes, and potentially reduce processing and communication costs. Decomposition of complex SMCs into simpler SMCs lengthens the information processing chain, thus increasing the chance of information sharing among multiple independent subscribers. Furthermore, SMC decompositions allow for more effective distribution of the SMCs, where localized (zero-communication cost) processing may be achieved.

4.4.1 Predicate Decompositions

An SMC predicate is a boolean expression, which may be decomposed using boolean algebra. These decompositions, however, are only effective if disjoint operands are produced (i.e. an SMC is decomposed into many SMCs, that hold mutually exclusive groups of QEs). Every SMC then either examines a distinct part of the overall condition or

Table 4: Decomposed Explosion SMC

(a) Temperature_High SMC

SMC	
N	"Temperature_High"
Q	$A := I(Temp).L(-30^r, 0^r).N.I(atomic);$
P^n	$A.value \geq 40$
P^x	$true$

(b) Explosion SMC

SMC	
N	"Explosion"
Q	$A := I(Light).N.N.I(atomic);$ $B := I(Sound).S.N.I(atomic);$ $C := I(Temperature\_High).I(0^r).N.I(atomic);$
P^n	$\lvert A.time - B.time \rvert \leq 0.5 \,\&\&\, C.valid$
P^x	$true$

joins the partial results to examine the overall condition. The *Explosion* SMC may be decomposed as in table 4.

The resultant SMCs may capture meaningful conditions, that can be shared with other high-level conditions.

4.4.2 QE Decompositions

Predicate decompositions can lead to the separation of QEs, but each individual QE may also be decomposed. This decomposition distributes the SK search-space over a number of SMCs, such that every SMC searches a disjoint table of the KP-combinations (cartesian product of KPs).

This decomposition only relates to the QEs that hold a $multiple$ parameter. The range of the parameter can be decomposed into an arbitrary number of disjoint ranges, each of which is examined by an independent SMC (e.g. the $L(-30^r, 0^r)$ parameter may be decomposed into $L(-30^r, -15^r)$ and $L(-15^r, 0^r)$).

The decomposed SMCs search only parts of the knowledge-space, hence their findings (SKs) are limited and incomplete. Events published by these SMCs need to be joined to examine sub-parameter conditions, and yield accurate results. This join operation depends on the chosen sub-parameter, and in its simplest case (for the $multiple : separate$) is not necessary. Other sub-parameters require appropriate join operations that assert their conditions over the realised SKs.

multiple:any requires a logical OR join operation to yield only a single result (SMC event) from the SKs.

multiple:all requires a logical AND operation to ensure all attribute values are satisfied across all the SKs.

multiple:one requires a logical XOR operation to ensure the uniqueness of the satisfied attribute value in SKs.

These join operations can be expressed using SMCs. Table 5 gives an example of this decomposition for the *Temperature_High* SMC (shown in table 4(a)).

5 Evaluation

We first examine the expressiveness of SMCs in the SPS framework; in particular, we discuss two drawbacks that are commonly associated with the Pub/Sub systems. In the second part, we highlight an application scenario, for which we process real sensor data to capture high-level conditions.

Table 5: Decomposed SMCs

(a) Temperature_High1 SMC

SMC	
N	"Temperature_High1"
Q	$A := I(Temp).L(-30^r, -15^r).N.I(atomic);$
P^n	$A.value \geq 40$
P^x	$true$

(b) Temperature_High2 SMC

SMC	
N	"Temperature_High2"
Q	$A := I(Temp).L(-15^r, 0^r).N.I(atomic);$
P^n	$A.value \geq 40$
P^x	$true$

(c) Temperature_High SMC

SMC	
N	"Temperature_High"
Q	$A := I(Temperature\_High1).I(0^r).N.I(atomic);$ $B := I(Temperature\_High2).I(0^r).N.I(atomic);$
P^n	$A.valid \,\&\&\, B.valid$
P^x	$true$

Table 6: Filtering SMCs

(a) *Temp_0.5Hz* SMC

SMC	
N	"Temp_0.5Hz"
Q	$A := I(Temp).I(0^r).N.I(atomic);$
P^n	$last.time \leq -2^r$
P^x	$true$

(b) *Temp_10%Change* SMC

SMC	
N	"Temp_10%Change"
Q	$A := I(Temp).I(0^r).N.I(atomic);$
P^n	$\lvert A.value - last.value \rvert \geq 0.10 * last.value$
P^x	$true$

The performance and scalability of the SPS framework is discussed within the context of the application scenario.

5.1 Expressiveness

5.1.1 Controlling the rate of events

Although Pub/Sub subscribers have no control over the rate of event publication, the need for this control is evident. Consider a temperature sensor that publishes temperature readings (events) every second. While this granularity is suited for some applications (e.g. fire breakout monitoring), it may be too fine-grained for others (e.g. daily temperature logging). Subscriber-asserted control over the rate of event publication is useful, particularly when communication is a scarce resource.

In SPS, a subscriber may define an SMC that filters events according to a custom specification. Table 6 shows two SMCs that limit the rate of events, that are delivered to the subscribers, either by time or by value.

Temp_0.5Hz SMC publishes the most recent *Temp* event every $2s$, maintaining a fixed $0.5Hz$ event publication rate. *Temp_10%Change* SMC passes an event when the *value* attribute has changed by more than 10% (relative to the previously passed event's *value*).

Table 7: SensorInUse SMC

SMC	
N	"SensorInUse"
Q	$A := N.I(0^r).I(0^r).N; B := L.I(0^r).I(0^r).N;$
P^n	$A.name > Temperature$
P^x	$!(B.name > Temperature)$

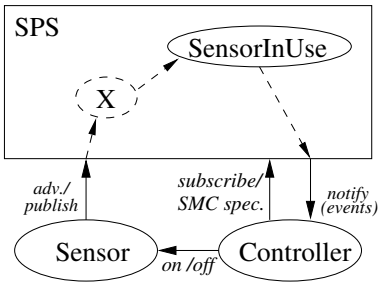

Figure 3: Controlling Sensors

5.1.2 On-demand event publication

One of Pub/Sub's downsides, for sensor networks, is its *push-based* communication model. This requires event publishers (e.g. sensors) to be actively sensing and publishing events that may be of no interest to event subscribers and subsequently discarded at the local event broker. Some related work has modified Pub/Sub implementations to allow event publishers to be shut down when no corresponding subscribers exist.

SPS's expressiveness rectifies this problem, without the need to modify the Pub/Sub implementation. Essentially, a feedback loop is created where the sensor's hardware-controller becomes a subscriber to a condition that detects interests about its sensor's data. As such, the controller can activate the sensor when interests arise, and de-activate it when interests perish, see figure 3.

We can detect if there exists any SMC, such as X in figure 3, that uses the local sensor (e.g. temperature) data. We assume that such an SMC publishes high-level (e.g. fire) events about our local environment. We can know about all SMCs that have advertised events about our local environment through the A QE in the *SensorInUse* SMC (table 7). This SMC examines for any SMC, such as X, that publishes high-level events which incorporate knowledge about the local sensor data. The entrance predicate is satisfied when an SMC like X exists, and the exit predicate is satisfied when no such SMC exists any more. The controller toggles the switch according to the received ingress/egress *SensorInUse* events.

5.2 Application Scenario: Journey Planning

The proposed framework has been implemented on Jist/Swans[3]. An application scenario was implemented and tested (using real-data from SCOOT[16]) to observe the scalability and performance of SPS.

Consider a smart transportation system, composed of different sensor devices, such as inductive loop sensors, speed cameras, ANPRs, and traffic light signals. In this system, users can benefit from high-level information to plan related activities. Let us define the following data sources for high-level information deduction in our experiment.

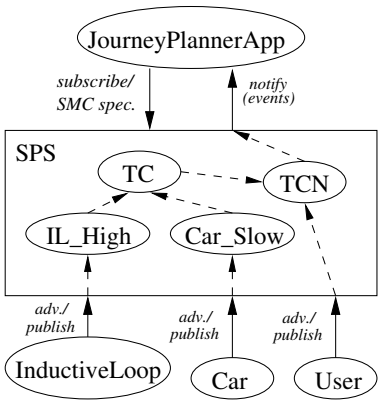

Figure 4: Application Overview

Table 8: TrafficCongestionNear SMC

SMC	
N	"TrafficCongestionNear"
Q	$A := I(TrafficCongestion).N(0^r, 0^r).S.N;$ $B := I(User).I(0^r).I(0^r).I(atomic);$
P^n	$\|A.location - B.location\| \leq 2$
P^x	$true$

- Inductive Loop sensors, with periodic reports on road-segment occupancies (continuous *InductiveLoop (IL)* event notifications at 1Hz).
- Speed Measurements[2], reporting on the speed of the passing cars on the road (discrete, but potentially high-volume, *Car (C)* event notifications).
- Location sensors, indicating the current location of the simulated users (continuous *User* events at 1Hz).

We decided to help mobile users to plan their journeys, by providing real-time traffic information about their nearby traffic congestions. More specifically, we simulated 500 mobile users, in a two-dimensional grid-like road network, that had subscribed to their nearby traffic congestions. The challenges in this application were two-fold.

Firstly to capture "traffic congestion" conditions, which we defined as the mutual occurrence of "high road occupancy" and "slow vehicle speeds". These were deduced from the inductive loop sensor data, and the speed camera readings. Every congestion lasted until the speed of the moving vehicles exceeds a given threshold value ($15MpH$).

Secondly, traffic congestion reports needed to be filtered, according to individual user locations. Since our input knowledge was confined to the published event notifications, we introduced *User* event notifications (detailed above) that were published periodically to provide location information about each individual user. Traffic congestion reports were filtered according to the most recent *User* event knowledge, which meant users did not need to update their subscriptions as they moved around the network.

The *TrafficCongestionNear* event topic, whose SMC is detailed in table 8, reports on the nearby traffic congestions, that are situated within a 2 road-junction distance of the user's present location. Users subscribed to this SMC locally, meaning that SMCs were hosted locally and filtered

[2]inferred from a secondary stream of raw SCOOT data

Table 9: Decomposed TrafficCongestion SMC

(a) *TrafficCongestion* SMC

SMC	
N	"TrafficCongestion"
Q	$A := I(IL\_High).I(0^r).S.I(ingress);$ $B := I(Car\_Slow).I(0^r).S.N;$
P^n	$A.valid \&\& B.valid \&\& A.location == B.location$
P^x	$(!B.valid) \&\& (B.location == last.location)$

(b) *InductiveLoop_High (IL_High)* SMC

SMC	
N	"IL_High"
Q	$A := I(InductiveLoop).A(avg, (-30^r, 0^r)).S.$ $I(atomic);$
P^n	$A.value > 2.5$
P^x	$A.value < 2.5 \&\& A.location == last.location$

(c) *Car_Slow* SMC

SMC	
N	"Car_Slow"
Q	$A := I(Car).A(avg, (-60^r, 0^r)).S.I(atomic);$
P^n	$A.value < 7$
P^x	$A.value > 15 \&\& A.location == last.location$

TrafficCongestion events for the local subscriber.

The manually decomposed version of the *TrafficCongestion* SMC, which captures traffic congestion conditions and publishes *TrafficCongestion* events is outlined in table 9. See also figure 4 for a high-level view of the system.

We examined sixty roads, each equipped with a single inductive loop and speed measuring sensor. In addition to these, 380 network nodes were introduced to ensure wireless network connectivity in the simulation environment. All nodes were assumed to have resources to house SMCs. Twenty hours of real data, relating to two distinct days, 1^{st} *July*, 2006, 1*AM*–9*PM* for Sim1, and 6^{th} *April*, 2006, 1*AM*–9*PM* for Sim2 were processed. Table 10 outlines the simulation parameters, and table 11 outlines the results.

Table 10: Simulation Parameters

Statistics *Annotations*	Sim1 *1/7/06*	Sim2 *6/4/06*
Number of SMCs and Clients		
Client – User	500	500
Client – InductiveLoop (IL)	60	60
Client – Car	60	60
SMC – IL_High	60	60
SMC – Car_Slow	60	60
SMC – TrafficCongestion (TC)	256	256
SMC – TrafficCongestionNear (TCN)	500	500
Client – JourneyPlannerApp (JPA)	500	500
Client Subscriptions *journey planner applications*	500	500
Resolved Subscriptions *(including InfoS Subscriptions)*	2132	2132
Max Subscriptions per node *InfoS subscriptions for TC and TCN SMCs*	2	2
Number of primitive Event Notifications		
User	3.6e+7	3.6e+7
Car	330736	494682
InductiveLoop	4.32e+6	4.32e+6

Table 11: Simulation Results

Statistics *Annotations*	Sim1 *1/7/06*	Sim2 *6/4/06*
Decomposed SMC types *IL_High, Car_Slow, TC and TCN*	4	4
Decomposed and Distributed SMCs	876	876
Max SMC allocation per node	1	1
Max SMCs observed per node *TC SMCs*	4	6
Max Events stored per InfoS *Car events*	37	30
Max Events stored per InfoS for the TC SMC evaluation *IL_High, Car_Slow, and TC events*	14	21
Max received Events for TC SMC	236	701
Max Predicate Evaluations per received Event Notification *TrafficCongestion predicate evaluations*	10	14
Event Publications by Clients *IL, Car, and User Events*	40650736	40814682
Total Event Notifications (ENs) *published Events (including SMC ENs)*	40656552	40825670
Events Disseminated (ED) *Events Disseminated within the Network*	1784	3884
Events Delivered to Subscribers *TCN Events to JPA Clients*	4032	7104
Number of high-level Event Notifications		
Car_Slow	774	1048
IL_High	842	2540
TrafficCongestion	168	296
TrafficCongestionNear	4032	7104

5.3 Performance

Our performance evaluation aimed at examining the effectiveness of SMCs, their decompositions and distributions in the context of a real application scenario. We were particularly interested in the following parameters, each of which was also related to the expressiveness, induced processing, communication, and storage costs of SPS.

- How many distinct SMC types were defined to support the outlined application scenario?
- How much load-balancing was achieved?
- How much localization was achieved?
- How much information sharing was attained?

The application scenario was fully described using four SMCs (shown in figure 4), three of which were localized (either fully or partially) at the publisher nodes. The *InductiveLoop_High* and *Car_Slow* SMCs were fully localized, meaning that they were automatically decomposed (along the QE location attributes) and distributed over the 60 *InductiveLoop* and 60 *Car* event publishers. The *TrafficCongestionNear* SMC was partially localized - located on the nodes that hosted *User* event publishers (i.e. the 500 mobile user nodes), but also received non-local events.

The above localizations meant that only *InductiveLoop_High*, *Car_Slow*, and *TrafficCongestion* events were disseminated across the network - other events were processed or consumed locally. The disseminated events totalled to 1784 (Sim1) and 3884 (Sim2).

With regard to load-balancing, although only four SMCs were defined, the number of SMCs, following autonomous decompositions and distributions, totalled 876. This included 620 SMCs that were localized at the publisher nodes (discussed above). With localization, the processing is also limited to the examination of locally generated data.

The original *TrafficCongestion* SMC was computationally expensive to evaluate. The cartesian product of the KPs, representing the most recent "high road occupancies" and "slow vehicle speeds", produced large data sets for examination. This process was repeated per receiving *InductiveLoop_High* or *Car_Slow* event at the *TrafficCongestion* SMC. The decomposition of the *TrafficCongestion* SMC, however, resulted in 256 SMCs that focused on smaller ($\frac{1}{16}$) regions of the environment. This decomposition reduced the maximum number of events that were received at any one SMC from 1616 (Sim1) and 3588 (Sim2) events (relating to a centralized SMC) to 236 and 701 events.

The *TrafficCongestion* SMC decomposition and distribution not only reduced the number of receiving event notifications, which in turn reduced the frequency of SMC evaluations, but also reduced the size of the knowledge that was maintained at the local InfoS components. The maximum observed processing complexity was $10n$ (Sim1) and $14n$ (Sim2), meaning that the maximum size of the KP-combinations relation (produced by the cartesian product of the KPs) were 10 and 14 tuples, respectively. This compares to $74n$ (Sim1) and $90n$ (Sim2), that would have been realised if the *TrafficCongestion* SMC was not decomposed. From the 876 distributed SMCs, 376 SMCs collaboratively deduced the traffic congestion information. The 168 (Sim1) and 296 (Sim2) *TrafficCongestion* events, published as result of this collaboration, were disseminated to the 500 *TrafficCongestionNear* SMCs that were localized at their mobile user nodes. This sharing meant that the processing, storage, and communication costs, that were associated with these 376 SMCs, were divided among the 500 users when examining per-user costs in the system. Thus, as the number of users increases, the per-user cost decreases towards the cost of disseminating *TrafficCongestion* events to a user, and filtering these events for nearby traffic congestion reports. This demonstrates the scalability of the SPS, that is achieved as a result of information sharing.

6 Conclusions

In this paper we focused on State Maintenance Components and how they may be used to capture expressive conditions in an SPS supported sensor system. We demonstrated how temporal and spatial condition interrelationships can be defined, and provided a fine-grained explosion detection example to accompany our discussions. Our evaluation, using real sensor data, demonstrated the effectiveness of SMC decomposition and distribution. This decomposition allowed for load-balancing and distributed data processing, as well as effective information sharing when interests overlapped.

Acknowledgements

This work was funded by *Microsoft Research Cambridge*.

References

[1] T. Abdelzaher, B. Blum, D. Evans, J. George, S. George, L. Gu, T. He, C. Huang, P. Nagaraddi, S. Son, P. Sorokin, J. Stankovic, and A. Wood. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In *Proc. of 24th International Conference on Distributed Computing Systems (ICDCS)*, Japan, March 2004.

[2] Asaf Adi and Opher Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2):177–203, 2004.

[3] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. Scalable wireless ad hoc network simulation. *Handbook on Theoretical and Algorithmic Aspect of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks*, pages 297–311, 2005.

[4] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[5] Ramesh Govindan, Joseph M. Hellerstein, Wei Hong, Sam Madden, Michael Franklin, and Scott Shenker. The sensor network as a database. Technical Report 02-771, USC/Information Sciences Institute, September 2002.

[6] Oliver Kasten and Kay Römer. Beyond event handlers: Programming wireless sensors with attributed state machines. In *The Fourth International Conference on Information Processing in Sensor Networks (IPSN)*, pages 45–52, Los Angeles, USA, April 2005.

[7] Shuoqi Li, Sang Hyuk Son, and John A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In Feng Zhao and Leonidas J. Guibas, editors, *IPSN*, volume 2634 of *Lecture Notes in Computer Science*, pages 502–517. Springer, 2003.

[8] Jie Liu, Maurice Chu, Juan Liu, James Reich, and Feng Zhao. State-centric programming for sensor-actuator network systems. *IEEE Pervasive Computing*, 02(4):50–62, Oct-Dec 2003.

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

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

[11] Multidimensional Expressions (MDX) Reference. http://msdn2.microsoft.com/en-gb/library/ms145506.aspx.

[12] D. Moreto and Markus Endler. Evaluating composite events using shared trees. *IEE Proceedings - Software*, 148(1):1–10, 2001.

[13] Peter R. Pietzuch, Brian Shand, and Jean Bacon. Composite event detection as a generic middleware extension. *IEEE Network*, 18(1):44–55, 2004.

[14] Kay Römer and Friedemann Mattern. Event-based systems for detecting real-world states with sensor networks: A critical analysis. In *DEST Workshop on Signal Processing in Sensor Networks at ISSNIP*, pages 389–395, Melbourne, Australia, December 2004.

[15] N. Sadagopan, B. Krishnamachari, and A.Helmy. The ACQUIRE mechanism for efficient querying in sensor networks. In *Proc. 1st Intl. Workshop on Sensor Network Protocols and Applications (SNPA)*, Anchorage, AK, May 2003.

[16] SCOOT. http://www.scoot-utc.com.

[17] Salman Taherian and Jean Bacon. SPS: A middleware for multi-user sensor systems. In *Proc. of the 5th workshop on Middleware for pervasive and ad-hoc computing*, pages 19–24, Newport Beach, USA, November 2007. ACM Press.

[18] Salman Taherian and Jean Bacon. State-filters for enhanced filtering in sensor-based publish/subscribe systems. In *Proc. of the 8th International Conference on Mobile Data Management (MDM'07)*, pages 346–350, Germany, May 2007.

[19] Y. Yao and J. Gehrke. The COUGAR approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.