Reactive Programming Optimizations in Pervasive Computing

Chao Chen, Yi Xu, Kun Li and Sumi Helal
Mobile and Pervasive Computing Laboratory
Computer & Information Science & Engineering Department
University of Florida, Gainesville, FL 326011, USA
{cchen,yixu,kli,helal}@cise.ufl.edu
www.icta.ufl.edu

***Abstract*—Pervasive computing systems are begging for programming models and methodologies specifically suited to the particular cyber-physical nature of these systems. Reactive (rule-based) programming is an attractive model to use due to its built-in safety features and intuitive application development style. Without careful optimization however, reactive programming engines could turn into monstrous power drains of the pervasive system and its sensor network. In this paper we propose two optimizations for reactivity engines. The first, which we prove to be optimal, assumes all sensors in the space are equally important to the application. The other, which is adaptive, employs and estimates a probability for each sensor based on application usage. Both optimizations use a mixed push/pull approach to achieve optimal or near optimal energy efficiency. We present an experimental evaluation of the two algorithms to quantify their performance over a range of parameters.**

Keywords-rule based processing; reactivity engines; programming models in pervasive spaces; optimization; perfromance

I. INTRODUCTION

Establishing new programming models especially suited for pervasive spaces is crucial to the effectiveness and robustness of the development process and the manageability of the space through its entire life cycle. Such new programming models are also needed to improve the programmers' productivity by defining the precise role and scope of the programmer within other roles specified by the model.

While many early research projects in pervasive computing highlighted novel applications, they typically required ad hoc programming efforts to create highly customized software. Such efforts were merely system integration efforts rendering rigid systems that are difficult to change or re-program.

Recently, more researchers started to explore various methodologies and concepts to enable and define more usable programming models for pervasive spaces. For example, [1] and [2] created a database abstraction for pervasive spaces which allows programmers to access physical objects and their attributes in a query-like syntax. While query is a powerful tool for retrieving information from sources such as sensor data streams, they lacked actuation support. The work in [3] and [4] utilized a context-based view of the pervasive space in which actions and behaviors of a system are directly driven by contexts. However, compared to query processing, the computational burden of context inference and reasoning raises critical implications of performance. Besides, neither model has addressed the issue of interoperability.

A more recent breed of programming paradigm [17] based on the Service-Oriented Architecture (SOA) has enabled a new programming environment allowing semi-automated integration and interaction among various system components, which are represented as services. The service-oriented device architecture (SODA) model focuses on the services provided, rather than the sensor data streams or active contexts of the environment. Services are stackable. So simple services can be composed together and form into more complex services.

We have developed a reference implementation of SODA (Figure 1.), which features the Atlas sensor platform – a platform similar to the Berkeley mote but different in its explicit support to externalized programming through the service oriented model. The reference implementation consists of the Atlas sensor platform and a two-pronged middleware known as the Atlas middleware. As such, our implementation combines hardware nodes (sensor platforms) and firmware running on the hardware, with a software middleware running in the network to provide services and an execution environment. Together these components allow virtually any kind of sensor, actuator or other more complex devices to be integrated as software services into a common data/services bus. Since the Atlas middleware enables service composition and gracefully handles situations when services are updated, created and aborted, a programmer simply needs to find the proper services and arrange the service method calls to create an application.

We have applied the Atlas service-oriented programming model in several research project developments including the Gator Tech Smart House (GTSH) [16], which is a real-life deployment of an ambient assisted living space. One lesson we learnt from our deployment in the GTSH is that SODA is a great model but has two critical limitations.

- *First,* SODA tends to overpromise the capabilities of services by overlooking and hiding their inherent limitations. While service interfaces abstract away unneeded information on internal details of the service, it also blinds the programmer from "seeing" critical information such as the physical limitations of a device, sensor or actuator. Therefore, the opacity of a service interface creates risks for service misuse that could lead to what we call "unsafe programming".
- *Second,* SODA encourages a free and unrestricted style of application composition over available device

services. While this is a powerful capability to the programmer, it may lead to unpredictable behaviors in the pervasive space when the applications are deployed. In theory, a programmer can select any set of services from the service pool and connect them to form an application. Service methods can therefore be invoked in any order. Clearly, such lack of constraints on service composition may unintentionally but adversely cause conflicts and contentions among the various services. This could create a potentially enormous state space of the system and opens up possibilities for states that are undesirable, unpredictable or even impermissible. It is therefore important to refine the SODA programming model into a more controlled and constrained model that does not compromise the overall system operation while taking advantage of the benefits of service orientation.

Based on the lessons learned and the SODA analysis briefly described above, we have developed a rule-based model extension to bring about control and intentional restrictions. We refer to this model as event-driven SODA programming model or E-SODA. In this model, application logics are represented by a set of rules, each of which follows a typical Event Condition Action (ECA) structure. By constantly checking on sensor readings and updating rule evaluations, the pervasive system listens to certain events in the space and responds to them by taking specific prescribed actions. Compared to pure SODA, this extension model enables a streamlined and constrained way for program logic formulation that is more stringent and less error-prone. In addition, event composition creates a tighter programming space than unrestricted service composition, reducing possibilities of false, non-intentionally erroneous, or impermissible executions in the pervasive space. Furthermore, the rule-based nature of this extension model encourages a centralized reasoning engine where conflicting applications logics can be easily detected and resolved.

For the E-SODA model to work effectively in pervasive spaces, its event processing must be optimized to take into account the limitation on energy use by the sensor nodes and the sensor network. If the model is applied without optimizations, the constant evaluation of events and rules would pose hefty computational burdens to the centralized data sinks. Additionally, the continuous data sampling by the sensor nodes and transmission through the network will incur substantial energy cost to the entire sensor network. It is therefore unthinkable to implement an event-driven model such as E-SODA without a framework of relaxation that provides meaningful opportunities for optimization to minimize both energy and computational cost.

In this paper, we present the Atlas Reactivity Engine (ARE), an implementation of the E-SODA event-driven programming model within the Atlas architecture. We briefly describe the Atlas architecture and show how the engine extends the original service-oriented programming model. Then we introduce two important components for rule composition: the ECA grammar structure and a Time-Frequency Modifier (TFM) operator that enables a per event relaxation of rule evaluation. The relaxation is intended to be specified by the programmer based on application and event semantics. We describe the concept of *optimal push-pull envelope* that guides our optimization approach, which includes a static optimization algorithm (OPT-1) as a base technique and an adaptive algorithm (OPT-2) that supports dynamic sampling rate changes and exploits the opportunity for short-cut evaluation. We prove the optimality of OPT-1 mathematically and show the performance gains of the OPT-2 algorithm through an experimental evaluation study.

II. REACTIVE PROGRAMMING USING ARE

A. The Atlas Architecture

As Figure 1 shows, the Atlas reference architecture [15][16][17] contains physical, sensor platform (node), service and application layers. At the physical layer, a variety of devices including sensors, actuators, and more complex devices are deployed to monitor and control the environment. All these devices are connected to the Atlas Sensor Platform nodes in the sensor network. The service-oriented Atlas sensor platform automatically integrates these devices and represents them as service bundles in the service layer. These bundles implement a uniform service interface that abstracts away most of the physical details and helps programmers to concentrate on the essential aspects of the services that an object provides. At the application layer, programmers can easily compose services and develop applications using a SOA-based programming IDE (utilizing the SODA model).

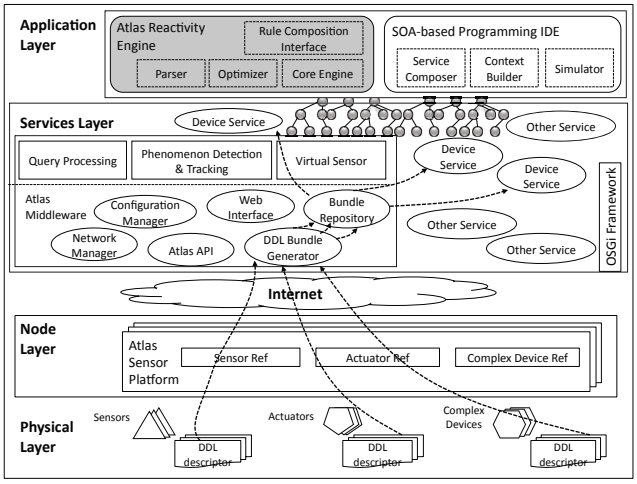

Figure 1. Atlas reference architecture featuring support for the SODA programming model (through SOA-based programming IDE – top right box) and for the E-SODA, event-driven extension model (through the Atlas Reactivity Engine - top-left, highlighted).

This paper describes the Atlas Reactivity Engine (ARE), which implements the E-SODA programming model. ARE's basic elements are rules defined over SODA services. By specifying the events, conditions and actions, programmers formulate rules that describe the desired/allowed behavior of the space in various situations. To ensure the adherence to these rules, the reactivity engine constantly checks sensor data and evaluates the rules. When certain events happen and

a rule evaluates to true, corresponding actions (e.g. actuating a device or invoking another service) will be taken to respond to the event.

B. ECA: Grammar for Rule Composition

ARE follows an Event/Condition/Action paradigm in which a set of rules are defined by the programmer, registered, maintained and appropriately triggered as the sensor data changes. ARE implements Events, Conditions, Actions, Rules and Commands, as follows:

1) EVENT: An event is a logical expression over sensor values. Formally, an event, E, is defined in equation (1). The event defined in the first line represents an atomic event, with *value* indicating the desired value of Sensor or [a, b] indicating a desired range of Sensor values. The event defined in the next line represents a composite event. The operator + is intended as logical *OR*. The operator *, which has higher precedence than +, is intended as logical *AND*. The operator *seconds* is a modified *AND* operator in which the concurrence requirement is relaxed. The right operand event is allowed to take place up to "seconds" number of seconds after the left operand event has occurred (is evaluated to true).

$$\begin{aligned} E &= Sensor(value) \mid Sensor[a,b] \\ E &= E + E \mid E * E \mid E * \text{seconds} * E \end{aligned} \tag{1}$$

2) CONDITION: Conditions are intended as logical expressions of variables local to the ARE. A condition, C, is defined as (2). The same condition may participate in more than one rule. Conditions are useful in debugging and are an added trigger guard which adds safety to the coding process.

$$C = \{True \mid False\} \tag{2}$$

3) ACTION: An action is an invocation of one or more methods belonging to one or several application or device service bundles. An action, A, is defined as:

$$\begin{aligned} A &= Service.method; \\ A &= (A; A) \end{aligned} \tag{3}$$

4) RULE: A rule is a specific configuration of a predefined event E, condition C, and action A, and is therefore defined as (4), which means if Event E happens, while condition C is true, Action A should be triggered.

$$RULE = \backslash\backslash\, E, C, A \,\backslash\backslash \tag{4}$$

C. TFM: a Time-domain Relaxation Operator

Since the working mechanism of ARE is rule-based, the rate at which rules are evaluated determines the traffic of sensor data within the sensor network. By default, each rule gets evaluated continually and constantly. Although, this default strategy guarantees responsiveness of the ARE engine, as mentioned before, it is obviously unpractical due to the heavy network traffic and sensor sampling.

Fortunately, application and event semantics offer good opportunities for optimization and partial evaluations. Semantically, some events are tolerant to time fidelity in that they do not require frequent evaluation at all times. For example, when monitoring room temperature, an air conditioning system does not need to get temperature reading at a very high frequency, since the rate of change of room temperature is slow. In addition, an air conditioning system may not need to monitor room temperature during spring and fall.

Considering the above scenarios, we introduce the notion of time/frequency modifier (*TFM),* a relaxation operator intended to assist the programmer in specifying time constraints or time relaxations in connections with *EVENT*. The *TFM* is specified as follows:

$$\begin{aligned} TFM &= < W, F_e > \\ W &= \text{nil} \mid \text{date/time-date/time} \mid \text{time-time} \\ F_e &= \#\ \text{of seconds between two successive evaluations} \\ \text{date} &= \text{MM/DD/YY} \\ \text{time} &= \text{hh:mm:ss} \end{aligned} \tag{5}$$

Where W is a time window in which the affected event needs to be evaluated with frequency F_e. To incorporate *TFM* into *EVENT*, the following modifier definition should be added to *EVENT*:

$$E = TFM(E) \tag{6}$$

The significance of *TFM* is that it introduces a per event time relaxation of event evaluation into ARE, which gives us the opportunity to optimize the engine to achieve energy efficiency.

D. Implementation

The overall architecture of ARE is shown in Figure 2. An ARE command interpreter facilitates input of commands from the programmer and output of confirmations and rule trigger notifications to the programmers. The ARE engine is a service bundle that registers new rules and their components. It also serves as a trigger mechanism for actions of "applicable" rules. It communicates with the Atlas middleware services on one hand and the command line interface service on the other hand. The ARE optimizer is at the heart of the engine, constantly guiding its decisions and operation. The "Rule Set" database contains all the rules defined by the programmer. At startup, a rule execution schedule table is generated by the engine. The actual execution is governed by the initially generated table, current state of the engine, and of course the optimizer and its algorithms.

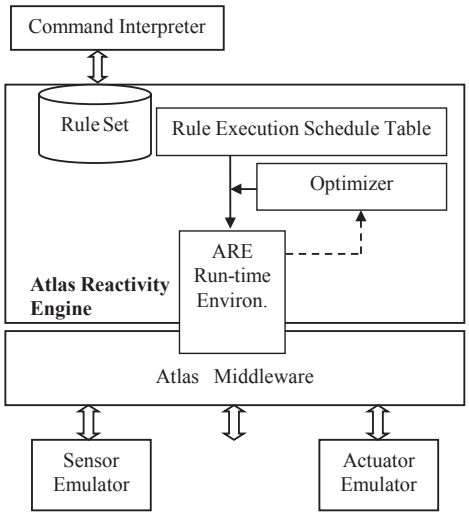

Figure 2. Overall architecture of ARE.

III. OPTIMIZING THE REACTIVITY ENGINE

A. The Push-pull envelope concept

To acquire fresh sensor readings for rule evaluation, ARE employs two alternative ways to communicate with the sensor network: push and pull. The push mechanism allows the engine to subscribe to a particular sensor for continuous readings at a constant rate, while the pull mechanism enables an on-demand style of data query to acquire readings one at time. Each mechanism could be advantageous or disadvantageous, based on the specific set of rules being executed by the engine. When sensor data are needed at a constant rate, pushing requires much less downlink traffic since the engine subscribes only once, while pulling pays the round-trip penalty for each data query. However, pushing loses its edge when handling sporadic data needs, as a subscribed sensor has to sample and transmit data even when they are not needed by the engine, leading to a substantial waste of energy in the sensor network.

To balance the tradeoffs between pushing and pulling, we propose a hybrid approach to achieve a near-optimal energy cost. We describe the core idea of this approach using the push-pull envelope concept (Figure 3.). A push-pull envelope is an optimal configuration of hybrid push/pull proportions for each sensor over the lifetime of execution of a group of rules. It effectively describes an engine execution whose combined cost of push and pulls is minimal. More specifically, we employ a strategy, which, by analyzing and predicting the patterns of sensor data required by the engine, could separate those constant and dominant data demands from the sporadic, transient ones. For each sensor, the push-pull envelope establishes an optimal base push rate to meet those dominant demands, and supplements it with reactive pulls to satisfy the rest of the demands. The combination of push and pull will reduce network traffic and the cost of sensor sampling (reading), and hence the overall energy consumption in the sensor network.

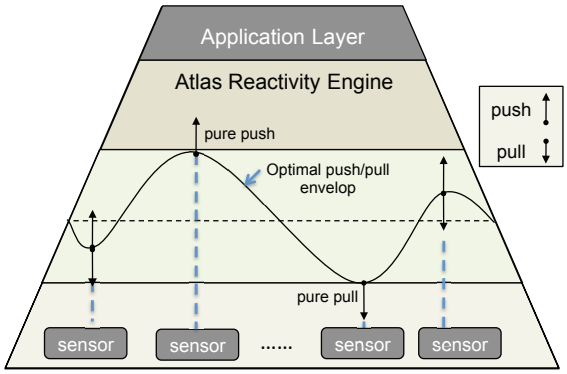

Figure 3. A push-pull envelope strikes the balance between pushing and pulling to reduce energy cost in the sensor network.

B. The base push alogrithm

This section describes a static algorithm that finds a constant base push rate that is optimal, denoted by f^*, for a single sensor. We observe that the minimal sampling rate required by the engine usually changes with time, depending on the different states of the pervasive system. To meet this varying requirement, the engine adjusts the rate of supplemental pulls to affect an overall (base + supplement) rate change. Therefore, finding the optimal f^* that minimizes the total energy cost is key to the success of the algorithm.

We assume all atomic events are equally probable to occur (*i.e.* evaluated to true), so the algorithm treats them fairly to compute f^*. The TFM operator partitions an event into various phases, each requiring a different sampling rate on the corresponding sensor. This partition goes further when the evaluation of multiple rules overlap, provided that these rules involve the same sensor. We define this partition as follows.

Definition 1: Let a sensor's timeline L be the length of the execution time of the engine. A partition P(L) divides L into n time windows, each window i is of length l_i ($1 < i \le n$) , and has a requirement of the minimal sampling rate f_i which is defined as:

$$f_i = \max\{f_i^k, k \in \mathbf{K}\}. \quad (7)$$

K is the set of all events associated with the sensor at time window i and f_i^k is the min sampling rate required by an event k.

The optimization problem is as follows. For a partition P(L), find the optimal base frequency f^* such that the objective function $C(f^*)$ is minimal. We construct the objective function by modeling the energy cost on a sensor node. In the cost model, we consider both transmission and sampling [5] as major factors contributing to the overall energy consumption, and define two energy cost coefficients accordingly. First coefficient is α which is the energy consumption factor for one time transmission (either sending or receiving a packet), while second coefficient is β which represents the energy cost for one sensor sampling (reading). Therefore, the cost of a pull operation is $2\alpha + \beta$ (receiving

query + sending data + sampling), and a push costs $\alpha + \beta$ (not including the one time subscription). The objective function is defined as:

$$C(f^*) = C_{push}(f^*) + C_{pull}(f^*) = \sum_i [(\alpha+\beta) * f^* + (2\alpha+\beta) * Q_i(f^*)] * l_i,$$
$$where\ Q_i(f^*) = \begin{cases} f_i - f^*, & if\ f_i > f^*, \\ 0, & otherwise. \end{cases} \quad (8)$$

The proposed algorithm can be summarized as follows:

OPT-1: Base Push Algorithm for Finding Optimal f^*
Input: for a sensor s, the partition P(L) and the min sampling rate f_i for each time window i.
Output: the base push rate f^*.
Algorithm:
1. Sort all time windows in the increasing order of f_i.
2. **for** k = 1, ..., n,
3. compute $g(k) = (\alpha+\beta)\sum_1^k l_i - (2\alpha+\beta)\sum_k^n l_i$
4. **if** $|g(k)| \le |g(k+1)|$
5. **return** f_k *as the optimal* f^*
6. **else**
7. **goto** 3.

The complexity of the algorithm is bounded by the sorting procedure in step 1, which is $\boldsymbol{O}(n\log n)$. To prove the correctness this algorithm, we first show that after step 1, the cost function can be rewritten as:

$$C(f^*) = \sum_1^k (\alpha+\beta) f^* l_i + \sum_{k+1}^n (2\alpha+\beta)(f_i - f^*) l_i \quad (9)$$

We compute the first derivative of $C(f^*)$, which is a discrete function of k:

$$C'(f^*) = g(k) = (\alpha+\beta)\sum_1^k l_i - (2\alpha+\beta)\sum_{k+1}^n l_i \quad (10)$$

$C(f^*)$ reaches its minimum point when $|g(k)|$ is closest to 0. We can show that $g(k)$ is monotonically increasing, and we are bound to find a k among [1, … n] such that $|g(k)|$ is closest to 0. Therefore the algorithm will always return an f^* that minimizes the cost function $C(f^*)$.

In this method, optimization is based on the assumption that, sub-events of a composite event have equal probability of occurrence. Therefore, all the sub-events are treated fairly when designing the algorithm of OPT-1to find the optimal f^*.

However, for a composite event, the probabilities of its sub-events are most likely to be different in most situations. In the next section, we propose a short cut approach, which takes advantage of the uneven distribution of probability of events, allowing rule evaluation to focus on a smaller set of sub-events, which, while not optimal, achieves a great deal of energy saving.

C. Adaptive Short Cut Approach

Consider the event grammar where a composite event takes the form: $E=E_1+E_2+...+E_n$. Obviously E evaluates to true if any one of its sub-events (e.g. E_1) is true. So there is no need to evaluate sub-events when one has already evaluated to true. Similarly, no need to evaluate sub-events when one has evaluated to false in * composite events ($E=E_1*E_2*...*E_n$). This inspires what we call the *short cut* optimization, a new technique, similar to what can be found in compiler expression optimization, applied in the reactivity engine to reduce the number of events to be evaluated, and hence the overall power consumption of the sensor network.

We assume that a composite event is composed of sub-events with different probability of occurrences. As we mentioned before, some of these sub-events may have significantly higher probability of occurrence than others which makes tham a dominant factor in determining the value of the composite event to which they belong. We call such sub-events *dominant sub-events*. Apparently, if we first evaluate the dominant sub-events of a complex event, it is highly probable that we may short cut further evaluations of other sub-events.

Motivated by this idea, we propose a probability-based approach, which takes advantage of the nature of dominant events and the benefit brought by short cut evaluation. We first find the dominant sub-events set S^*. Whenever an event needs to be evaluated, sub-events in S^* get the highest prorioty. Herein, push mechanism is adopted to supply sensor data for event evaluation given that push incurs less energy cost than pull mechanism does knowing that the engine does require the pushed data. Other sub-events will get evaluated only when S^* fails to determine the value of the event. Because this could only be determined during execution time, ARE will pull data from the sensor if additional sensor readings are needed to complete the event evaluation. Consequenly, finding the set of S^* will help reduce the number of events to be evaluated. However, if S^* is not suitable, ARE may pay penalty for large number of supplemenantal pulls which is more costly in terms of energy consumption and ultimately may bring down the overall performance and energy efficiency. To sum up, our goal is to find the proper set S^* that leads to the minimized energy cost.

To implement the idea of short cut, we look into two problems.

First, finding S^* depends on our knowledge of probability of event. Thus, one thing we have to do a priori is to find or estimate the probability of all the events. Let p_e represent the probability of event e. In our model, we estimate p_e as the proportion of times event e is evaluated to be true. As shown in equation (10), oc_e denotes the number of times event e occurs (i.e. event e is evaluated to be true) and ev_e represents the number of times event e is evaluated. In addition, oc_e and ev_e are repeatedly observed throughout the entire execution period.

$$p_e = \frac{oc_e}{ev_e} \quad (11)$$

The second problem that we need to solve to enable the short cut idea is to capture the variability in the probability of

events over time. That is, S^* that satisfies one rule evaluation requirement may not be suitable for subsequent evaluations. An adaptive approach should therefore be provided to adjust sensor's push and pull rate to adapt to this change. This also requires that the sampling rate of the physical sensors be dynamically reconfigured during execution time, which is the case in Atlas. Thus, finding a suitable algorithm for adaptation is needed.

Before describing our algorithm, we first define the cost function that helps identify the set of S^* leading to the optimal solution. Suppose we have a set of rules R, a set of sub-events S, which is composed of all the sub-events that appear in R and a set S_i which is composed of sub-events that appear in R_i.

$$R = \{R_1, R_2, R_3, ..., R_m\}$$
$$S = \{E_1, E_2, E_3, ..., E_n\} \quad (12)$$

Assuming that probability of each event in S is already known. Then the expectation of the total energy consumption is a function of S^* as shown below:

For rule R_i :

$$C^i_{push} = (\alpha + \beta) * (\sum_{e \in S^* \cap S_i} f_e)$$
$$C^i_{pull} = (2\alpha + \beta) * q_i * (\sum_{e \in (S - S^*) \cap S_i} f_e)$$
$$C^i = C^i_{push} + C^i_{pull}$$

Then the total energy cost is :

$$C(S^*) = \sum_{i=1}^{m} C^i \quad (13)$$

In this function, f_e denotes the minimal frequency of evaluation that is required by event e. C^i denotes the total energy cost by evaluating rule R_i. C^i includes two parts: C^i_{push} which represents the energy cost because of the usage of push mechanism and C^i_{pull} which represents the energy cost due to the usage of supplemental pulls. $(\alpha+\beta)$ is the average energy cost of pushing a sensor reading to ARE, whereas α, β have already been defined in OPT-1. $(2\alpha+\beta)$ is the average energy cost of pulling data from a sensor by ARE. q_i denotes the probability that pull mechanism is adopted for evaluating rule R_i (i.e. the sub-events in S^* fail to determine the value of the composite event). q_i is calculated as following:

$$q_i = \prod_{e \in S^* \cap S_i} (1 - p_e) \quad (14)$$

Our goal now is to find the optimal S^* that minimizes the total energy consumption cost $C(S^*)$. A brut force approach is to try all possible combinations of events in S and find the combination, which yields the lowest energy cost. However, such enumeration is obviously exponential with complexity $O(2^n)$. In addition to exponential enumeration, S^* needs to be modulated repeatedly throughout the whole evaluation phase due to the dynamic nature by which event probabilities are estimated. To tame this complexity we propose a greedy algorithm to help find a near-optimal set S^* which confines the complexity within $O(n^2)$.

OPT-2: Greedy Algorithm for Finding S^*
Input: Rule set R, sub-event set S, probability of each sub-event in S
Output: S*, minimal energy cost c
Algorithm:
1. $S^*=\Phi$; $c=C(S^*)$
2. **for** $e_i \in (S - S^*)$,
3. compute $c_i = C(S^* \cup \{e_i\})$.
4. Find the event e_j that yields the minimal cost c_j;
5. **if** $c_j < c$,
6. $S^* = S^* \cup \{e_j\}$, $c = c_j$, **goto** 2.
7. **else** return S^* and c

As has been stated before, the energy cost may degrade over time as the probability of event p_e changes. To adapt to this change, we adopt a tolerance threshold γ that represents the largest deviation between current cost and expected cost the algorithm could bear. S^* is recalculated whenever current cost is out of the range $[c^*(1-\gamma), c^*(1+\gamma)]$, where c is the minimal cost returned by the algorithm of OPT-2.

Although this greedy algorithm cannot ensure optimal solution, we do guarantee that each time when we increase the size of S^* by one extra event, we can always have a lower energy cost.

IV. Experimental Evaluation

A. Experimental Setup

We evaluate the performance of the Atlas reactivity engine (ARE) under OPT-1 and OPT-2 algorithms, by comparing their performance with that of ARE under pure push and pure pull mechanisms. We use an emulation approach where actual Atlas middleware and ARE are used, but in which Atlas hardware nodes are replaced by software emulators. The Atlas middleware and ARE have both been developed in Java and runs under the Knopflerfish OSGi, version 1.3.4.

We used 50 Atlas device emulators, each representing either a sensor or an actuator. In the case of sensors, each emulator generates a sensor reading every 2 seconds. The readings are randomly generated using a Gaussian distribution function whose mean changes with time at a step length of 8. We have designed several test cases (or rule sets), which were randomly generated using a fixed rule structure. Each rule in a rule set included one complex event with a fixed length of 5 sub-events. Each event is tagged with a TFM relaxation of a randomly generated frequency and time window. To create rule sets of variable size, we generated a large pool of rules and carefully constructed the desired rule sets from this pool. We observed that ARE starts to thrash at 50 rules which seems to be the engine's current maximum capacity. Hence we generated rule sets of size {5, 10, 15, 20, 25, 30, 35, 40, 45, and 50}. We emulated the effect of three sets of sensor platforms (Table 1) by

normalizing their energy consumption coefficients. We conduct experiments on each set of coefficients to quantify the platform's receptiveness to our optimizations.

TABLE I. NORMALIZED ENERGY COEFFICIENTS FOR TRANSMISSION AND SAMPLING ON THREE DIFFERENT SENSOR PLATFORMS.

Sensor platform	α (transmission)	β (sampling)
S1	0.63	0.37
S2	0.03	0.97
S3	0.56	0.44

S1: MICA2 sensor platform with Sensirion Humidity sensor, ChipCon CC1000 Radio [20].

S2: Atlas sensor platoform with Interlink Pressure Sensor, Atmel ZLink Radio [5].

S3: MicroLEAP with ECG sensor, class-2 Bluetooth 2.0 radio [22].

B. Evaluation Metrics

The main performance metric, which is measured throughout all experiments, is the total energy consumption of the entire engine. Other metrics not reported in this paper is startup overhead (time) and runtime overhead (time) of the ARE engine. The actual total energy cost measured in all experiments is given by (15).

$$n_{push}(\alpha + \beta) + n_{pull}(2\alpha + \beta) \quad (15)$$

C. Performance Comparison Results

We compared the performance of OPT-1, OPT-2, pure push and pure pull for the three sensor platforms shown in Table 1 above. Energy cost was measured for a varying workload of rules (from 5 to 50 rules). Figure 4. shows the energy cost of S1 (MICA 2) under the four algorithms.

Results in Figure 4. show pure push to be the worst performer as expected. Both OPT-1 and OPT-2 outperform pure pull. With 50 rules running, OPT-1 saves 17% of the energy cost compared to pure pull, while OPT-2 achieves an additional 20% savings. Figure 5. shows the push-pull envelope generated by each optimization. In OPT-1, push makes the most part of the envelope. With the adaptive short-cut technique enabled, the number of push has significantly reduced, which results in a more balanced envelope between push and pull in OPT-2.

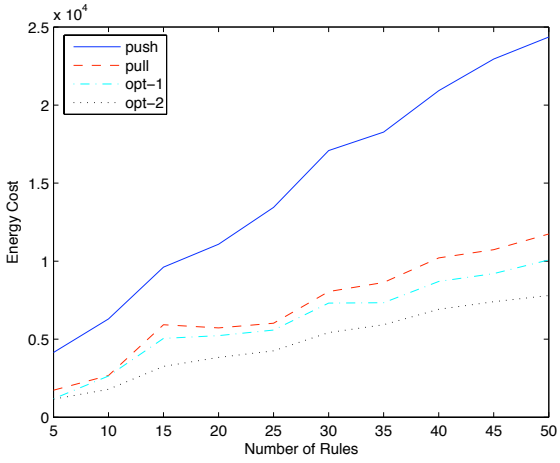

Figure 4. Energy cost for four algorithms on the MICA 2 sensor platform (Test Set A).

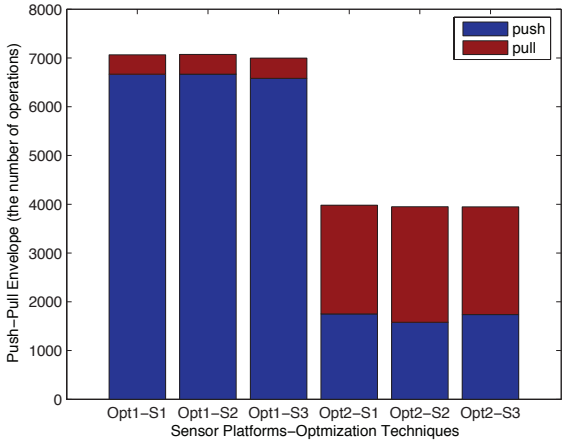

Figure 5. The push-pull envelope on various sensor platforms (S1-S3).

D. Platform Receptiveness to Optimizations

We emulate the effects of optimization on three sets of sensor platforms and compare the results in Figure 6. . We calculate the relative energy efficiency by comparing the optimized results with the cost of pure push (which is normalized to 1). While all three platforms receive substantial energy reduction, the Atlas sensor platform (S2) sees the most energy savings because of its low transmission co-efficient α. The optimization encourages pull on this platform due to the minimal communication overhead, and as a result, develops an envelope with a larger proportion of pulls (Figure 5. Opt2-S2).

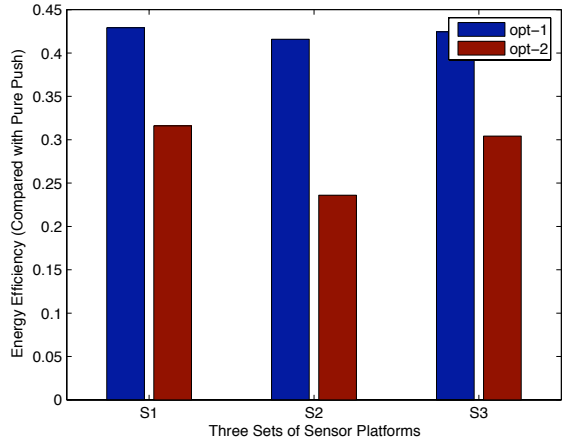

Figure 6. Energy efficiency for two optimization algorithms on three sets of sensor platforms.

E. Effect of the TFM Relaxation

To examine the effect of TFM relaxations on our algorithms, we compared the results of two test sets -Test Set A shown in Figure 4. and Test Set B shown in Figure 7. , each featuring a different average F_e in TFM. We designed set A with a measured mean F_e of 4.67 per second, drawn from a uniform distribution over [2.00-8.00] to model the

case of drastic change of sampling requirement. Set B, however, was designed with a measured mean F_e of 14.67 per second, representing a more moderate change of sampling requirement. The observed and measured effect of TFM can be summarized as follows. The total energy cost decreases in general for all four algorithms when a moderate change of sampling rate was used (Test Set B). Also, the difference between pure push and pure pull becomes less significant at moderate change rate. Also, unlike Test Set A (drastic sampling change rate), which encourages pull more than push, Test Set B strikes a better balance between push and pull. As a result, both push-based OPT algorithms performed better (as compared to pure pull). Finally, we observe that when frequencies decrease, OPT-2 shows difficulty in learning the probability at run time since there may not be enough samples at a moderate rate of change. This affects the algorithm's adaptability. Therefore, OPT-2 did not perform significantly better than OPT-1, as in Test Set A.

V. Related Work

Event-driven programming is becoming a popular paradigm of choice in system development for various application domains including healthcare [6], civil engineering [7] and security surveillance [8]. Extensions and modifications to the original event-driven model mainly focus on two areas: *event formulation* and *event detection*. A natural extension [18] to the event formulation structure is to support composite events. Two different mechanisms were proposed in [12] and [19] to manage the complexity caused by composite events in large-scale distributed systems. However, these approaches did not take into consideration any time relaxations that the events and the applications may be tolerant to. In our work presented in this paper, we introduce, justify, and take advantage of such time relaxation (the Time/Frequency Modifier relaxation). Furthermore, our approach takes advantage of *short cut* type of evaluation to further reduce the workload of the reactivity engine.

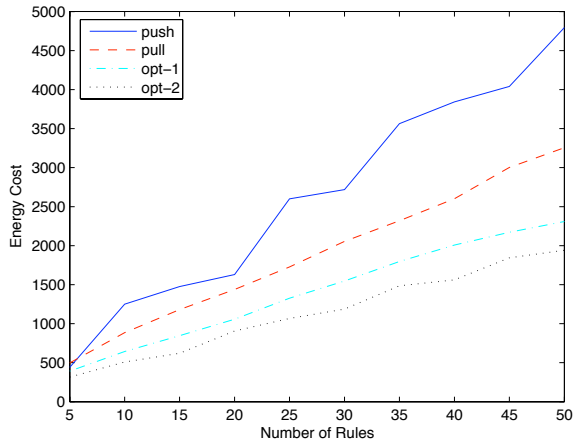

Figure 7. Energy cost for four algorithms on the MICA 2 sensor platform (Test Set B).

Another related research is event detection primarily focused on data acquisition techniques, especially in sensor networks. A typical scheme is polling [11], in which application sequentially polls its underlying sensors for new data. In contrast, a bottom-up sensor-driven model [12] has also been proposed, assuming that sensors are capable of pushing data to applications when event occurs. To improve the efficiency of data delivery and enable data sharing, messaging paradigms such as publish/subscribe [9] and push-pull [10] are widely adopted in sensor data acquisition. To improve the efficiency of event detection, a detection protocol over the publish/subscribe paradigm [20] has been proposed. Optimization techniques to balance push and pull have been extensively discussed in [10][13][14]. However, all of the above approaches focus on the network topology and routing algorithm, ignoring the optimization opportunity provided by the event structure and its time-frequency relaxations.

VI. Conclusion

We presented the Atlas Reactivity Engine, an optimized implementation of E-SODA – the event-driven programming model over the service-oriented Atlas middleware. We introduced a time-frequency relaxation of event execution creates (TFM), which we have shown to offer a rich opportunity for optimization. We presented the static power-aware base push algorithm (OPT-1) utilizing the concept of optimal push-pull envelope and proved its optimality. Based on OPT-1, we developed OPT-2, a predictive-corrective algorithm that adaptively modulates the push-pull envelope at runtime based on changing event probabilities as well as optimization opportunities exploited by TFM and the structure of the events. We conducted an emulation based evaluation study and quantified the significant energy savings due to OPT-1 and OPT-2, and compared the achieved savings with the pure push and pure pull cases.

References

[1] D. Massaguer, S Mehrotra, N. Venkatasubramanian. “A Semantic Approach for Building Pervasive Spaces”. Proceedings of the 6th Middleware Doctoral Symposium (MDS’09). November 2009.

[2] W. Xue and Q. Luo. “Action-Oriented Query Processing for Pervasive Computing”. Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR’05), 2005.

[3] T. Gu, H. K. Pung, J. K. Yao. “Towards a Flexible Service Discovery”. Elsevier Journal of Network and Computer Applications (JNCA). Vol. 28, Issue 3, pp. 233-248, May 2005.

[4] H. Yang, J. King, A. Helal and E. Jansen, “A Context-Driven Programming Model for Pervasive Spaces,” Proceedings of the 5th International Conference on Smart Homes and Health Telematics (ICOST), Nara, Japan, 21-23 June, 2007.

[5] R. Bose and A. Helal, "Sensor-aware Adaptive Push-Pull Query Processing in Wireless Sensor Networks," Submitted to the 6th International Conference on Intelligent Environments - IE'10, Kuala Lumpur, Malaysia, July 19-22, 2010.

[6] R. Fitterer, B. de Witte. “Enabling Pervasive Healthcare by Means of Event-Driven Service-Oriented Architectures - The Case of Bed Management in Mid-Sized to Large-Sized Hospitals”. Pervasive Health 2009.

[7] S. Jevtic, M. Kotowsky, R. P. Dick, P. A. Dinda, C. Dowding. “Lucid dreaming: reliable analog event detection for energy-constrained applications”. IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks, April, 2007.

[8] A. Boukerche, R. W. N. Pazzi, R. B. Araujo. "A fast and reliable protocol for wireless sensor networks in critical conditions monitoring applications". MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems. October 2004.

[9] N. Rosa, C.Ferraz, J. Kelner, E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira. "Mires: a publish/subscribe middleware for sensor networks". Personal and Ubiquitous Computing , Volume 10 Issue 1, December 2005. Springer-Verlag.

[10] X. Liu, Q. Huang, Y. Zhang. "Balancing Push and Pull for Efficient Information Discovery in Large-Scale Sensor Networks". IEEE Transactions on Mobile Computing, Volume 6, Issue 3, March 2007 Page(s):241 – 251.

[11] Z. Zhang, M. Ma, Y. Yang. "Energy Efficient Multi-Hop Polling in Clusters of Two-Layered Heterogeneous Sensor Networks".IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), April 2005.

[12] S. Reilly and M. Haahr. “Extending the Event-based programming model to support Sensor-Driven Ubiquitous Computing Applications”. Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications (Percom’09).

[13] Z. Tao, Z. Gong, Z. OuYang, J. Xu. "Two New Push-Pull Balanced Data Dissemination Algorithms for Any-Type Queries in Large-Scale Wireless Sensor Networks". International Symposium on Parallel Architectures, Algorithms, and Networks, 2008. 7-9 May 2008 pp: 111 – 117.

[14] S. A. Hashish, A. Karmouch. "Topology-based on-board data dissemination approach for sensor network." Proceedings of the 5th ACM international workshop on Mobility management and wireless access. Chania, Crete Island, Greece, 2007. pp. 33 – 41.

[15] J. King, R. Bose, H. Yang, S. Pickles and A. Helal, “Atlas – A Service-Oriented Sensor Platform,” Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006). Tampa, Florida, November 2006.

[16] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen, "Gator Tech Smart House: A Programmable Pervasive Space", IEEE Computer magazine, March 2005, pp 64-74.

[17] C. Chen and A. Helal, "Device Integration in SODA using the Device Description Language," Proceedings of the IEEE/IPSJ Symposium on Applications and the Internet, July 2009, Seattle, Washington, USA.

[18] P. Pietzuch, B. Shand and J. Bacon, "Composite Event Detection as a Generic Middleware Extension," Network, IEEE, vol. 18, pp. 44-55, Jan/Feb 2004.

[19] G. Starovic, V. Cahill, and B. Tangney, “An event based object model for distributed programming,” in Proceedings of the 1995 International Conference on Object Oriented Information Systems (J. Murphy and B. Stone, eds.), pp. 72–86, Springer-Verlag, December 1995.

[20] S. Lai, J. Cao, Y. Zheng, “PSWare: a Publish/Subscribe Middleware Supporting Composite Event in Wireless Sensor Network,”. IEEE International Conference on Pervasive Computing and Communications, 2009. pp.1-6.

[21] SR. Madden, MJ. Franklin, JM. Hellerstein, and W. Hong. “TinyDB: An acquisitional query processing system for sensor networks”. ACM Transactions on Database Systems, 30(1): 122-173, 2005.

[22] L.K. Au, W.H. Wu, M.A. Batalin, D.H. McIntire and W.J. Kaiser, “MicroLEAP: Energy-awareWirelessSensor Platformfor Biomedical SensingApplications”. IEEE BIOCAS2007. November 2007. pp.158-162.