Sensors **2013**, *13*, 2945-2966; doi:10.3390/s130302945

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

Explicit Context Matching in Content-Based Publish/Subscribe Systems

Sergio Vavassori [1], Javier Soriano [1], David Lizcano [2,*] and Miguel Jiménez [1]

[1] School of Computer Science, Universidad Politécnica de Madrid, 28660-Boadilla del Monte, Madrid, Spain; E-Mails: svavassori@conwet.com (S.V.); jsoriano@fi.upm.es (J.S.); mjimenez@conwet.com (M.J.)

[2] School of Computer Science, Open University of Madrid (UDIMA), 28400-Collado Villalba, Madrid, Spain

* Author to whom correspondence should be addressed; E-Mail: david.lizcano@udima.es; Tel.: +34-9020-20003.

Received: 25 December 2012; in revised form: 21 February 2013 / Accepted: 22 February 2013 / Published: 1 March 2013

Abstract: Although context could be exploited to improve performance, elasticity and adaptation in most distributed systems that adopt the publish/subscribe (P/S) communication model, only a few researchers have focused on the area of context-aware matching in P/S systems and have explored its implications in domains with highly dynamic context like wireless sensor networks (WSNs) and IoT-enabled applications. Most adopted P/S models are context agnostic or do not differentiate context from the other application data. In this article, we present a novel context-aware P/S model. SilboPS manages context explicitly, focusing on the minimization of network overhead in domains with recurrent context changes related, for example, to mobile ad hoc networks (MANETs). Our approach represents a solution that helps to efficiently share and use sensor data coming from ubiquitous WSNs across a plethora of applications intent on using these data to build context awareness. Specifically, we empirically demonstrate that decoupling a subscription from the changing context in which it is produced and leveraging contextual scoping in the filtering process notably reduces (un)subscription cost per node, while improving the global performance/throughput of the network of brokers without altering the cost of SIENA-like topology changes.

Keywords: content-based publish/subscribe; context-awareness; contextual scoping; overlay network

1. Introduction

Despite potentially featuring large numbers of nodes, most sensor networks currently work as isolated islands, and most of the sensed valuable data is not yet shared. In the last few years, some attempts have been made to share data with the global community, but they focus on sharing across multiple wireless sensor networks (WSNs), spatially deployed in different locations (*i.e.*, bridging WSNs on the Internet). Indeed, most of this research comes under the Large-Scale Wireless Sensor Networks umbrella [1]; it continues to address application-specific, static-sensor deployments, and does not tackle sensed-data sharing across different applications that could exploit the data outside the sensor network. Unless sensed data are shared across different application domains, the most important feature of ubiquitous computing—namely context awareness—will not easily reach its full potential.

With the advent of the Internet of Things (IoT), we can envision ubiquitous next-generation sensor networks that are connected to the Internet for publishing, sharing and searching sensed data across a wider range of IoT-enabled applications. The availability of advanced middleware functionalities to gather, process, exchange and exploit such data on a massive scale will be the cornerstone of the development of smart (customized, personalized and enriched) context-aware applications and services outperforming any available on the current Internet [2], thus fostering the creation of new business models and opportunities.

There is therefore an urgent need for a comprehensive solution that helps to share and use sensed data coming from ubiquitous WSNs across a plethora of applications intent on using these data to build context awareness. IoT-enabled cloud infrastructures, which are evolving to support ubiquitous and context-aware computation and information integration, could then leverage valuable sensed data to enable distributed applications to take into account the situation and context in which the information is produced or consumed. For example, services providing data could restrict the dissemination of their outputs to certain consumers based on their context, and any entity could subscribe to information whose provider matches a contextual scope.

The publish/subscribe (P/S) model in general, and its content-based form in particular, is poised as one of the most plausible approaches for achieving this goal, insofar as, traditionally, it has been the communication paradigm of choice for most WSNs. Moreover, the publisher and/or subscriber context (e.g., location information, environmental data, operating data, user preferences, *etc*.) could, if available, be relevant metadata for the routing process in a large fraction of the application domains in which the P/S model has been adopted as a smart solution for spreading information across a sizeable group of users or applications. This means that context has to be added to the P/S model and shared by publishers and subscribers.

In the IoT-enabled Internet, both generic data and context elements (*i.e.*, sensed data) are available for consumption through the same P/S interface. This will require a novel approach to the P/S

model that explicitly deals with sensed data (*i.e.*, context) as a separate, first-class component to coherently offer context-awareness. In short, context has to be natively added to the classic P/S model, *i.e.*, considered explicitly in the routing and forwarding processes and shared by publishers and subscribers. Unfortunately, despite many promising proposals [3–5], a powerful, fully-fledged context-aware P/S model is yet to be created. Precisely, the research community has predominantly focused on the wireless sensor network (WSN) field, where sensed data can arguably be regarded as context [6,7], but none of the most widely adopted content-based publish/subscribe (CBPS) middleware coherently offers context-awareness, as we will see in the analysis of current solutions. For example, it is not unusual to find context merely encoded into published notifications and subscriptions and handled as generic data, an approach which, as we argue in this paper, leads to major inefficiencies. Context-awareness radically changes the way both routing and matching needs to be performed in CBPS middleware to achieve efficiency. There are two orthogonal aspects in CBPS: *matching* and *routing*. *Matching* matches messages minimizing a metric, usually time, whereas routing routes the messages to other network elements, where the metrics are distance and energy [8–10].

In this article, we present a novel context-aware CBPS model, SilboPS, in which the context is managed explicitly. The focus is on the minimization of network overhead by improving the matching algorithm in domains with recurrent context changes related, for example, to WSNs, MANETs and IoT-enabled applications. Examples of contexts that have high or varying, "bursty" update rates include inventorying, stock portfolios, people or vehicle locators and proximity networks. We then evaluate our solution and compare it to SIENA [11,12], which is generally considered the reference implementation for a scalable CBPS service with a relatively low reconfiguration cost [13,14], and designed to maximize both expressiveness and scalability. Specifically, we empirically demonstrate the following:

Decoupling a subscription from the changing context in which it is produced and leveraging contextual scoping in the filtering and routing processes in CA-CBPS systems notably reduces (un)subscription cost per node, while improving the global performance/throughput of the network of brokers without altering the cost of SIENA-like topology changes.

Section 2, in particular, reviews the solutions and proposals of context-aware P/S models. Section 3 presents a use case highlighting the shortcomings of current approaches. In Section 4, we describe a set of design principles conceived to deal with the above issues. Section 5 further details our design decisions for implementing the proposed design principles. Next, Section 6 explains how our model behaves in comparison with SIENA in the proposed scenario and provide evidences supporting the affirmation stated above. Finally, in Sections 7 and 8, we conclude our proposal, and present future work in this area.

2. Background

As we are exploring how to exploit context to improve P/S systems as common mechanism for sensor networks, this section will briefly introduce key aspects of context management and, especially, context-aware CBPS systems.

Context-aware models are usually defined by how they react to context changes: *passively* or *actively* [15]. When passive models learn about a context change, they simply store the context or prompt the user before applying any change, whereas active models manage changes without user interaction,

enabling automatic contextual reconfiguration. Another way of classifying these models is by how a context-aware application realizes the context has changed, *i.e.*, either by *sensing* the environment or by *being notified*. Notification has the advantage of reducing communication overhead but at the risk of missing changes.

Syntactically, context can be transmitted using different message structures and syntaxes. The most widespread structures and syntaxes are in increasing order of complexity [2]: key-value pairs, markup scheme models, graphical models, object-oriented models, logic-based models, and ontology-based models. The flexibility/meaning trade-off differs from one representation to another. Key-value pairs are the best option for integration with CBPS systems since it is their canonical representation and, at the same time, has a rich internal representation for context modeling.

To deal with context [16] in CBPS systems, context information has to be processed from the viewpoints of both the subscriber and the publisher. Subscriber contextual information provides for filtering according to user location, device, preferences, *etc*. in order to get relevant, useful and appropriate information within the context. Publisher contextual information provides for message adaptation according to location or other context factors to get relevant, useful and appropriate information. Routing algorithms should bridge the gap, while leveraging both context scopes to reduce network overhead.

Context-awareness is a recent research line in the CBPS field, where most proposals are simply context agnostic. Within the context-aware approaches, most of the work has focused on implicitly context-aware strategies [17], which do not differentiate the context from the rest of application data and simply piggyback context on notifications or communicate context at subscription time. The shortcomings of these approaches are discussed in Section 3.

Most recent approaches have explored explicit context representation through an extended API to set the context that the broker network uses for notification routing and topology shaping [7]. However, these approaches have context-coupled subscriptions since both context and content filters are entangled making difficult them to manage independently.

Several middleware solutions have been designed to transparently implement context management and provisioning. The core assumption in [18] is that only effective and efficient context data distribution can pave the way to the deployment of truly context-aware services. This assumption is the basis for a unified architectural model and a new taxonomy for context data distribution. Similarly, a high-level software architecture for context data management and distribution suitable for m-commerce applications is presented in [19], but there is no detailed analysis of how the matching and the routing algorithms could be improved by using such contextual information. The comparison with other architectures is based on response time instead of throughput, being the latter the focus of this article.

Regarding the consideration of context in the filtering process, a major shortcoming of existing approaches and techniques is that they are very inefficient if profiles refer to values of context entity attributes that are subject to frequent changes (context updates). Recent research proposes the use of context-aware information filters (CIF) in addition to traditional information filters routing the input stream of messages in order to manage an input stream of context updates that are relevant to the routing process because the routing rules (a.k.a. profiles) refer to values in a context data store.

For example, [20] emphasizes the idea that information filtering systems and, therefore, their corresponding indexes must be context-aware and present AGILE (Adaptive Generic Indexing with Local Escalations), a set of extensions to existing index structures for information filtering systems aimed at dynamically adapting the index structure to the frequency of context changes. The key idea of AGILE is to dynamically adjust internal structure depending on the messages received, changing scope and accuracy accordingly. AGILE focuses on adaptive index structure management and applies to traditional database workloads, e.g., transaction processing and the TPC-C benchmark. Performance experiments showed that AGILE can improve the message throughput of a context-aware information filter by as much as one order of magnitude, compared with traditional approaches to implementing information filtering systems (No Index, Eager full indexing, Partial indexing, and Lazy Updates, GBU). Nevertheless, AGILE does not separate the context function from the subscription. Although this allows their matching function to outperform ours when the context changes slowly compared with the notification rate, it is a shortcoming when compared with our solution in highly changing context scenarios, as we will explain later.

Finally, systems can vary in terms of the aspects considered as context information. Most mature projects largely equate context with location management and are known as location-based services (LBS). L-ToPSS [21] is a LBS system based on a P/S middleware that adds an extra location processing module to a typical event broker to manage possible spatial events and subscriptions. The system is designed to support window queries and N-nearest queries. Based on L-ToPSS, Xu and Jacobsen [22] propose efficient algorithms for location constraint evaluation. CAMEL [23] is a push-based middleware construct based on a database. Like L-ToPSS, the system is designed to support window queries and N-nearest queries. Most of the research investigates how the constraints or predicates in subscriptions can be evaluated more efficiently, but little progress has been made with respect to enhancing the expressiveness of spatial subscriptions [24].

3. Shortcomings of Implicit Context Management

Taking into account roles, notifications and subscriptions, the problem is necessarily strongly affected by the entity context, in particular, their location. A nave application of CBPS to such a scenario would be to merge content and context in both messages and filters, making it implicitly context-aware. Although this paper focuses on the matching problem, an in-depth analysis unveils several shortcomings both related to the routing and the matching problem that are discussed in this section.

- **Matching inversion.** Classic CBPS systems model messages as key-value sets, $m \in \{\mathbb{K} \times \mathbb{V}\}$, where $\mathbb{K}$ is the attribute name and $\mathbb{V}$ is its value. Those messages can be matched by filters issued by subscribers, $f \in F \subseteq \{\mathbb{K} \times \mathbb{V}\}$. If m is in the scope of a filter f, we can state that f covers m or $f \succeq m$. The purpose of the message is to encode data, whereas the filter represents the interests in this information. Applying CBPS to context-aware scenarios violates this separation of concerns: subscriber filters contain constraints over the context mixed with constraints over the notification; symmetrically published notifications include constraints for subscribers that are enforced by brokers. As a result, there is the potential of the message and context being confused and semantically ambiguous. By mixing context and content, the application misses out on the

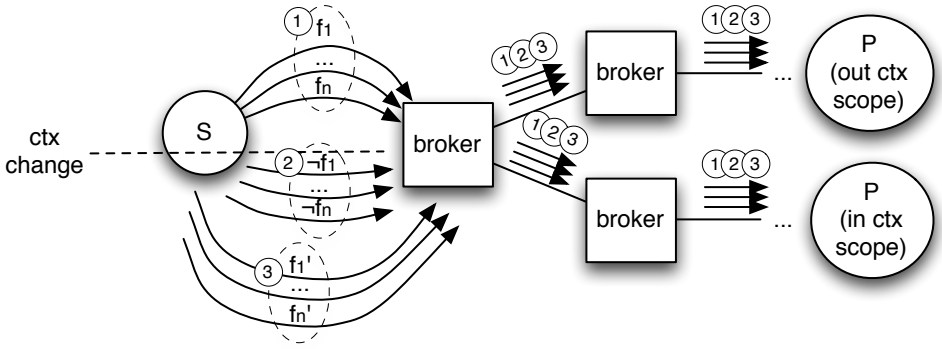
benefits stated in [25]. In addition, this can lead to *aliasing errors* if mixed context-aware and context-unaware entities use the same P/S system.

Only a few CBPS systems, which implement filtering using Turing-complete languages, can afford this out-of-the-box change. Anyway, such systems are difficult to optimize and exploit in the first place.

- **Message flooding in changing environments.** Published messages bundle the notification content with contextual information ($\mathbb{K} = \mathbb{K}_{content} \cap \mathbb{K}_{ctx}$), for example, the events detected by cameras include the camera location. This approach has a comparatively small overhead with respect to a context-agnostic scenario since the rate of change of the context is small compared with the rate of publication.

  If as in Figure 1 that assumption is not met, constant context updates will create a massive overhead because existing subscription filters (①) are constantly invalidated (②) and then updated (③). In terms of messages, this implies a waterfall of unsubscriptions and new updated subscriptions flooding the whole network as illustrated. In our scenario, driver subscriptions pose this problem since their position changes constantly, and they are encoded as part of the filter.

Figure 1. Exchange of messages with implicit context management.



- **Unrealized potential performance.** If properly exploited, contextual information can improve efficiency by minimizing networking overhead in two senses: (a) messages could be efficiently routed; (b) the overlay network topology could be optimized. For instance, it is of no use to cluster drivers interested in *nearby traffic alerts* (same interest) without considering the context that permeates their interest. Similarly, clustering drivers at nearby locations (same context) is plagued by the same problem, since it ignores their interests.

  Using implicit context, however, the broker will have to handle more complexity in the form of bigger routing tables and longer routing computation. Finally, any advantage from context exploitation requires explicit context management. Implicit context-aware solutions require the broker to parse both messages and filters, which is an additional overhead.
- **Separation of concerns within entities.** Context management and subscription or publication are very different concerns that must be handled by different components at the architectural level. Implicit context-awareness favors coupling and forces rigid designs in which the P/S component is also responsible for detecting, managing and communicating context changes.

For instance, drivers can use a mobile application in which location context is managed by a GPS component decoupled from the subscriber component in charge of handling incoming notifications.

- **Undermined publisher/subscriber decoupling.** The main property of CBPS systems is decoupling in terms of space, time and synchronization between publisher and subscriber. The only thing that the subscriber needs to know is how the events of interest are represented.
  Subscribers and publishers are obliged to share a unified and homogeneous representation of context, as context is embedded in notifications for implicitly context-aware systems. In the example scenario, location is represented uniformly as a coordinate point for all entities. A more flexible approach, such as ontology-based mediation, is beyond the scope of the scheme.

4. Proposed Design Principles

There is a clear motivation to overcome the weaknesses of implicit context-aware management, which degrade as the application scenario context becomes more changeable.

Our approach is to decouple context and content throughout the CBPS with the aim of spatially, temporally and synchronously decoupling publishers from subscribers within a *contextual scope*.

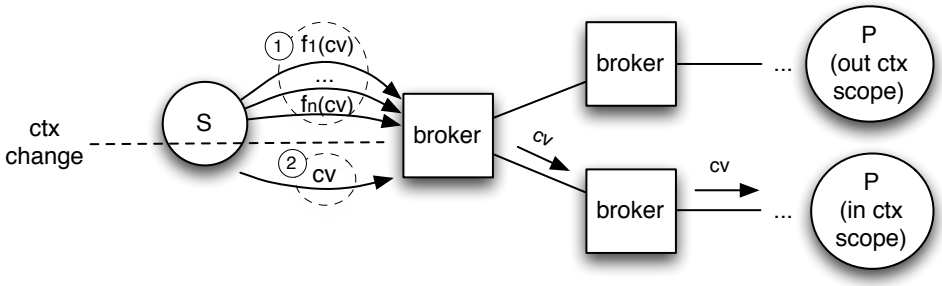
The design principles that guide our vision of a context-aware content-based publish/subscribe (CA-CBPS) system are as follows:

1. *Explicit separation of context and content.* The scope of both messages and their filters must be restricted and deal exclusively with content. Therefore, $m \in \{\mathbb{K}_{content} \times \mathbb{V}\}$.
2. *Generalized P/S model for contextual scoping.* The reversal of the matching procedure demonstrates an impedance mismatch between content-based filtering and any other type of filtering that does not fit the content-based model. Symmetric context-scoping filters will be attached to both subscriptions and publications, restricting publisher contexts and subscriber contexts, respectively. Such filters will be attached ($\|$) to publications, $m\|f_{cs}$, $f_{cs} \in F_{ctx}$, and to subscriptions $f\|f_{cp}$, $f_{cp} \in F_{ctx}$ given $F_{ctx} \subseteq \{\mathbb{K}_{context} \times \mathbb{V}\}$. Note how publisher and subscriber can restrict their actions symmetrically through each other's contextual scope. This scoping mechanism can be seen as a role-based access control by the publisher [26] and as a new filtering dimension by the subscriber.
3. *Context-invariant subscriptions and advertisements.* Subscriptions will be context-invariant to avoid context change flooding. Context variable references will be used instead of embedding context values (cv) as part of the filters. In fact, a subscription f_s defined as $f\|f_{cp}$ can be regarded as a function on the context in which the filter f_{cp} is the context-aware part:

$$f_s : \mathbb{C} \rightarrow F$$
$$f_s(ctx) \subseteq \{\mathbb{K} \times \mathbb{V}\}$$

Such functions become regular filters when applied to a given context $ctx \in \mathbb{C}$ within the brokers. Figure 2 illustrates how flooding is avoided as subscriptions (①) are updated by propagating context updates (②).

Figure 2. Exchange of messages with our proposed explicit context management.

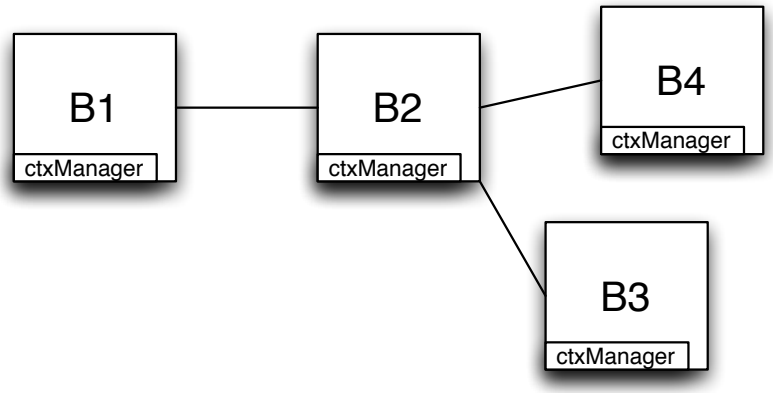


To apply context update and bind it to its correct filters, a subscriber-unique UUID, created by the broker, is added to filters and to context messages. This is necessary to distinguish the same filter if issued by different subscribers and to bind the filter to its correct context when the context function is called. This does not clash with the decoupling offered by P/S systems since it is used only internally by brokers and is comparable to the procedure enacted by other systems that use a multipath topology to resolve the duplicate message problem [27,28].

5. Proof of Concept: Proposed CA-CBPS Applied to the Use Case

This section shows the application of our proposal to the domain problem presented in Section 3. Our scenario includes a set of cameras sending traffic information (publishers) and a set of cars receiving relevant (context-aware) information (subscribers). Cameras are automatically connected to the brokers thanks to the infrastructure in which they are installed whereas cars can connect to brokers using the access points deployed along the road. This is an instance of a V2I scenario and is a simple sub-problem covering all the relevant details. At the same time this example fits the paper assumptions about environments with highly changeable context. Let us examine the state of the routing tables and the messages exchanged when adding publishers and subscribers to the network. Figure 3 shows the network of brokers, each of which has its own *contextManager*.

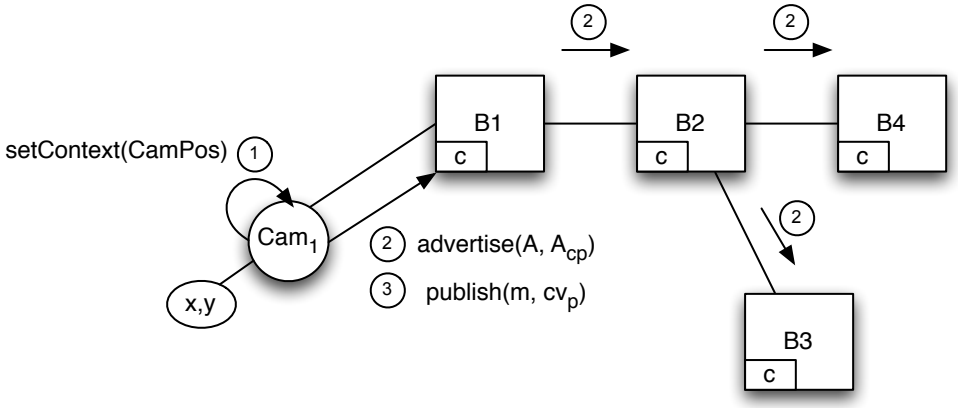
Figure 3. Scenario broker network.



5.1. Adding Publishers with Changing Context

A camera is connected to the broker network, specifically to broker B_1 whose *advertisement* and *subscription* tables are initially empty (see Figure 4). This camera establishes its initial context, namely its position, by invoking $setContext(CamPos)$ with $CamPos \equiv cv_p = \langle x_p = CamPos.x, y_p = CamPos.y \rangle$. However context updates from Cam_1 will not be sent to the broker since their value is meaningful only when bound to a notification. Instead the last context will be held by the local API and appended to the sending message on each *publish* method invocation. As a matter of fact, our assumption in this scenario is that context changes are more frequent than notification/subscription messages.

Figure 4. Adding a new publisher to the scenario.



Cam_1 is a publisher and, as such, must send an advertisement declaring what content and context data it can provide. This information will be useful for finding out what kind of information each entity provides and what contextual variables can be used to contextualize the messages received. Cam_1 sends $advertise(A, A_{cp})$, where:

- A is an advertisement filter as in traditional CBPS systems. Cam_1 can notify accidents: $A \equiv \{msg = accident\}$.
- A_{cp} models the context of the publisher, x_p and y_p, and shows what contextual information could be exploited by subscribers to enrich their subscriptions. Cam_1 has a location context: $A_{cp} \equiv \{\exists x_p, \exists y_p\}$.

This advertisement is propagated throughout the broker network to B_2, B_3 and B_4, producing changes in all their advertisement tables:

- B_1, *advertisement* table: $\langle Cam_1; A, A_{cp} \rangle$
- B_2, *advertisement* table: $\langle B_1; A, A_{cp} \rangle$
- B_3, *advertisement* table: $\langle B_2; A, A_{cp} \rangle$
- B_4, *advertisement* table: $\langle B_2; A, A_{cp} \rangle$

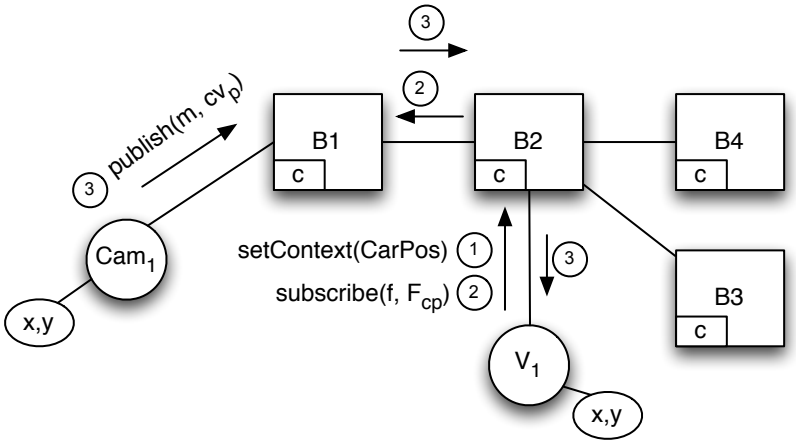
Cam_1 can now send notifications to the P/S infrastructure. B_1 receives $publish(m, cxt)$, where m is the notification, which is context-free content, and cxt, is the last context sent by Cam_1. B_1 queries the *subscription* table to forward the message. Since it is empty, B_1 drops the message m until it receives

a subscription following the path marked by previous advertisements. If the traffic camera is a mobile sensor and changes its position, it will use *setContext* to set the new position, but this will not be sent to broker B_1 until *publish* is invoked. Although for this scenario the context could be directly passed at publication time, interaction is cleaner if the API offers two different methods because there is no need for clients (publishers or subscribers) to remember the last context.

5.2. Adding Subscribers with Changing Contexts

Now a first-time subscriber joins the scenario (as in Figure 5). The subscriber is a vehicle, V_1, using a context-aware mobile device to receive alerts created within a 20 km radius of its position, which, for simplicity's sake, we model as a square with 40 km sides. Initially, $V_1's$ location will be in the range of Cam_1.

Figure 5. Adding a new subscriber to the scenario.



First of all, V_1 accesses the P/S network as a client of B_2 and sends its initial context, $setContext(CarPos)$. B_2 receives this information as a *context message*, extracts the cv_s values and creates a UUID (V_1^{id}) for V_1, which it stores to its *contextManager* table as $\langle V_1^{id}; x_s = CarPos.x, y_s = CarPos.y\rangle$. Then B_2 checks, using the *subscription* table, to see if V_1 context needs to be forwarded. In this case, no action is taken since the table is empty.

Next, B_2 receives the subscription of V_1 to nearby traffic alerts, $subscribe(f, f_{cp})$, which is mandatory since it has sent a $setContext(CarPos)$ call, where f is a content-based filter $f \equiv \{msg = accident\}$ and f_{cp} is a function that models a filter based on restrictions about publisher context, and receives dynamic subscriber context values:

$$\begin{aligned} f_{cp} \equiv \{&x_p \geq this.x - 20 \\ &x_p \leq this.x + 20 \\ &y_p \geq this.y - 20 \\ &y_p \leq this.y + 20\} \end{aligned}$$

The broker first adds V_1^{id} to the new subscription f', which is then stored into B_2's *subscription* table as $\langle V_1; f', f_{cp}\rangle$. The next step is to check if there is any advertisement that matches the subscription. To

do this, B_2 queries its *advertisement* table, which contains row $\langle B_1; A, A_{cp}\rangle$ and then checks if there exists an $A \succeq f' \wedge A_{cp} \succeq f_{cp}$. Given that A and f contain $msg = accident$ and that A_{cp} is $\exists x_p \wedge \exists y_p$ and f_{cp} uses exactly x_p and y_p as the publisher context, the match is positive, and B_2 sends first the context message, if needed, and then the subscription message to B_1. To send the context message, B_2 checks whether the *subscription* table already contains an outbound entry for V_1^{id}. If it does not, it sends the context message, namely $context(cv_s)$ with cv_s the *context value* containing $\langle V_1^{id}; x_s = CarPos.x,\ y_s = CarPos.y\rangle$, to B_1. Then it sends a subscription message with $subscribe'(f, f_{cp})$.

When B_1 receives both messages, it adds the subscriber's context cv_s to the *contextManager* as $\langle V_1^{id}, cv_s\rangle$ and modifies its *subscription* table with the subscriber's data, writing row $\langle B_2; f', f_{cp}\rangle$. B_2 and B_1 then stop propagating the subscription because there are no more entries in their *advertisement* tables.

5.3. Notification Delivery

To illustrate how new publications are to be delivered to interested subscribers, imagine that there is an accident close to Cam_1. Cam_1 sends a new $publish(m, cv_p)$, where $m \equiv \{msg = accident\}$. The publication includes the contextual information because the previous *setContext* called by publisher did not actually send the changes to the broker.

B_1's *subscription* table contains row $\langle B_2; f', f_{cp}\rangle$ and therefore B_1 must check if the publish message matches these filters, dropping it otherwise. Because $m \succeq f$, B_1 must check the contextual filter. This is a lazy process led by usually more restrictive and relevant content filters.

At this point, the context-invariant filter f'_{cp}, which is the key component of our proposal, must be evaluated using the values fetched from the *contextManager* table. In particular, B_1 uses V_1^{id} that searches f' to look up context information of the associated $cv'_s \equiv \langle x_s = CarPos.x',\ y_s = CarPos.y'\rangle$ and then passes both context (cv_s and cv_p) to evaluate f'_{cp}:

$$\begin{aligned} f'_{cp}(cv_p, cv_s) = f''_{cp} \equiv \{&CamPos.x \geq CarPos.x - 20 \\ &CamPos.x \leq CarPos.x + 20 \\ &CamPos.y \geq CarPos.y - 20 \\ &CamPos.y \leq CarPos.y + 20\} \end{aligned}$$

Note that the use of contextual values to evaluate context-aware restrictions has been delayed until now, improving performance in scenarios like this where context changes (car positions) occur at a faster rate than new notifications (traffic accidents). In addition, the context function is more expressive than the one available in subscriptions since it is able to perform simple computations and is not confined only to comparison.

In this case, the car is close enough to the camera, and the result of evaluating f'_{cp} is a positive match. Since both filters are satisfied, the message is sent to B_2 as the *subscription* table prescribes. To do so, $B1$ generates an internal publish message $publish'(m, cv_p)$ with the contextual variables of the publisher used in the previous step, namely $cv_p \equiv \{x_p = CamPos.x,\ y_p = CamPos.y\}$. Note again that the contextual data, in this case information about the publisher, is not propagated until strictly necessary.

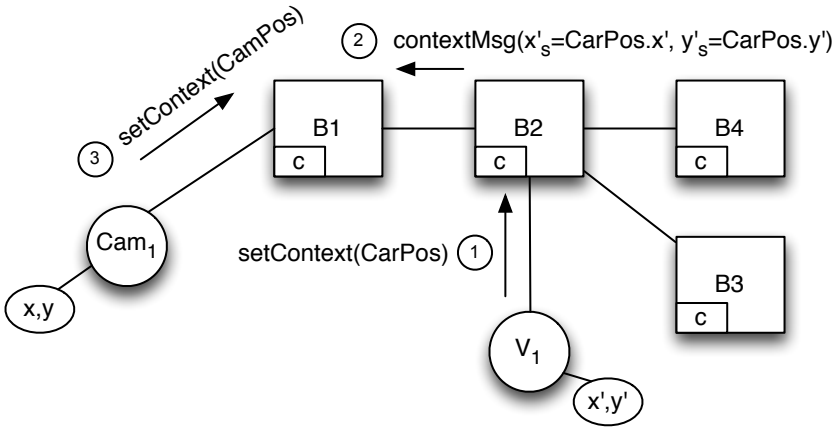
This publication reaches B_2. First, B_2 accesses the *subscription* table and checks that the publication matches row $\langle V_1; f', f_{cp} \rangle$. This process is identical to the above: $m \geq f$ so f'_{cp} must be evaluated as $f'_{cp}(cv_p, cv_s) \rightarrow f''_{cp}$. B_2 ultimately obtains the same contextual filter, and so will still match.

As the publication matches V_1's range, the message m carrying information about the nearby traffic incident is finally delivered to car V_1.

5.4. Subscriber Context Update

First, let us consider that V_1 is driving along the highway and changes its context (see Figure 6) by sending *setContext*(*CarPos*). B_2 processes this change of context with the help of the context manager, which replaces the values of x_s and y_s previously stored in the *contextManager* table.

Figure 6. Context change of a subscriber.



B_2 receives the new context values $cv'_s \equiv \langle x_s = CarPos.x', \ y_s = CarPos.y' \rangle$, creates the V_1^{id} and checks if it is different from the last value entered in the *contextManager* table. If it is, it updates the table with the new values, then looks up the outbound entries for V_1^{id} used to send the context message in the *subscription* table and finds B_1. If the new and old values are the same, no action is taken. Then B_1 receives this context message and modifies its *contextManager* table accordingly; next it queries the *subscription* tables, repeating the process carried out by B_2.

Subsequent publications by Cam_1 will be subject to a new contextual match, using the new data. f''_{cp} may no longer match, thus no messages will reach B_2. This way, tables are updated elsewhere in the broker network for every context they receive.

5.5. Publisher Context Update

Now camera position changes. Again, Cam_1 invokes the local API with *setContext*(*CamPos*) but no message is sent to the broker; instead the new values $cv'_p \equiv \langle x_s = CamPos.x', \ y_s = CamPos.y' \rangle$ are held locally and sent together with the following *publish* call.

6. Prototype Evaluation

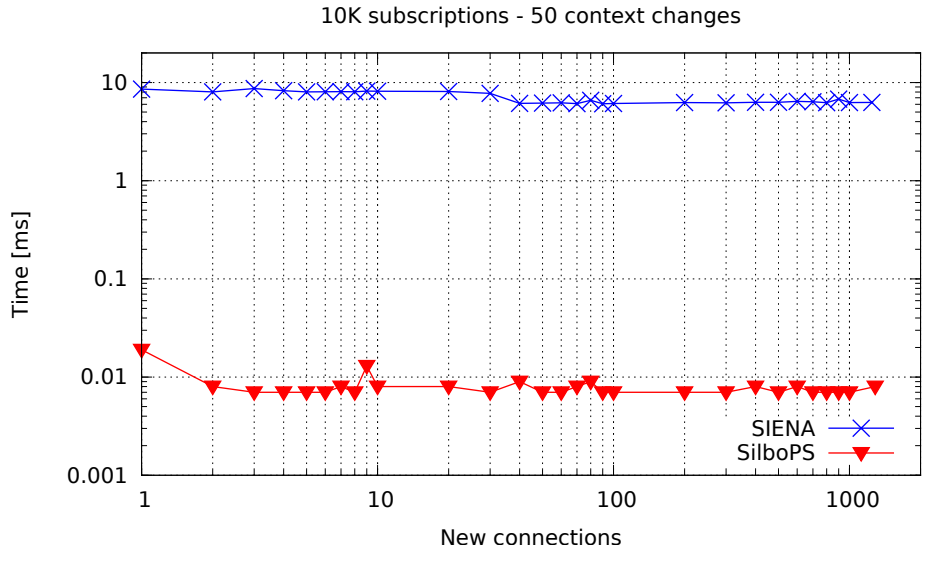
SilboPS evolved from our Java implementation of SIENA [12] and extends its interface by adding context functionality. In this way we were able to test the two systems in the same environment and match their differences without incurring in any semantic mismatch API penalization. As usual for P/S systems, our prototype does not support transactional messages, but offers FIFO channels with reliable delivery. This implies that no message loss will be observed unless there is a link failure; however both channel endpoints are aware of such failure. Messages are processed in the same order as they arrive at broker irrespectively of their sending time.

We ran our experiment under Linux 3.2 and OpenJDK 7u3 on an Intel® i7 860 @ 2.80 GHz machine equipped with 4 GB of RAM. Due to our scenario assumption, the focus is on context reconfiguration cost, rather than notification delivery performance. Nevertheless we ran a notification delivery performance test to verify the overhead introduced by this extension.

To assure that our results were meaningful, we used the same distribution parameters as in [12], adding the context part as an attribute of notifications and subscriptions for the SIENA version, and as a context function/context message in SilboPS. We used a uniform distribution applied on a square with 1,000 km sides to generate position values. The choice of this distribution gives SIENA a slight edge since it can exclude filters that do not match the notification position value, whereas SilboPS has to wait the context function result.

Thus, a context change in SIENA requires two messages to be sent: one message to remove the old filter, if any, and the second message to add the new filter. This is necessary because otherwise subscriptions will pile up leading to poor router performance and, above all, the delivery of unwanted messages.

Figure 7. Time normalized on connection to perform 50 context change on each new subscriber.



The first experiment is meant to compare the context change cost for the two systems. It performs 50 changes per subscriber on a single broker previously loaded with 10,000 subscriptions (equivalent

to 1250 connections). As shown in Figure 7, the operation has a fixed cost in both systems. However, SilboPS outperforms SIENA by three orders of magnitude. The reason is that SIENA has to modify its index entries (removing and adding the filters), instead of updating only the subscriber's context, whereas SilboPS only updates context values.

Figure 8 shows the total time taken by our prototype to perform context changes when it is loaded with an increasing number of connections. The router is loaded with a maximum of 10,000 subscriptions in the top chart and a maximum of 100,000 in the bottom chart. Each connection has an average of 10 subscriptions, that is, a load of 1,200 and 12,000 connections per router respectively. This highlights how processing time is proportional to the number of context changes and depends on neither the number of connections nor the number of subscriptions.

Figure 8. Time taken to change context with 10,000 and 100,000 subscriptions on a single broker in our prototype.

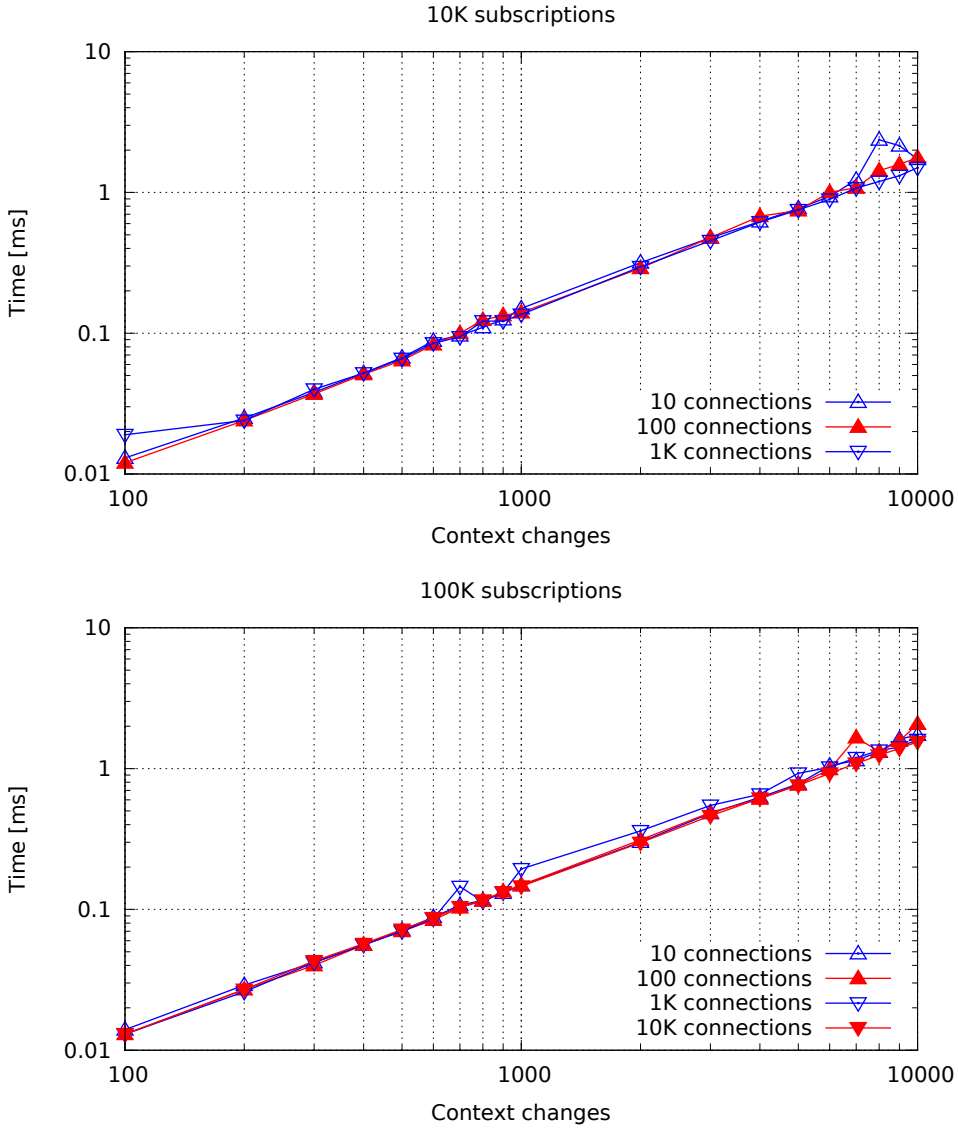


Figure 9 shows the cost of one context change with respect to the number of subscriptions issued by a single subscriber. It stands to reason that the cost increases linearly for SIENA, whereas it is constant in SilboPS. In addition to a faster processing time, this will reduce the number of messages sent in the network by a linear factor. So, the more subscriptions that are loaded to the broker, the bigger the saving will be. As a matter of fact, SilboPS behaves like SIENA when there is only one subscription per subscriber, but, even so, has the advantage of sending one instead of two messages, and the message size is smaller.

Note that the data in Figures 7 and 9 are consistent: the time required for a single context change in SIENA at 50 subscriptions per connection is equal to the value shown in Figure 7, that is, 10 ms. In this test SilboPS performance for a single context change appeared to be equal to its performance rate for 50 context changes. This is because, even though it has nanosecond precision, the available API does not guarantee nanosecond accuracy (As described in [29]). Even so we were able to calculate the slope for SIENA version and to get an estimate for the cost of a single context change of 105 µs/msg. For SilboPS we can either use Figure 7 where the time is high enough to be measured accurately, or use Figure 8 to calculate the slope, which is a fairly straight line. Thus we estimated an upper-bound of 0.2 µs/msg.

Figure 9. Time for a single context change with an increasing number of subscription per connection.

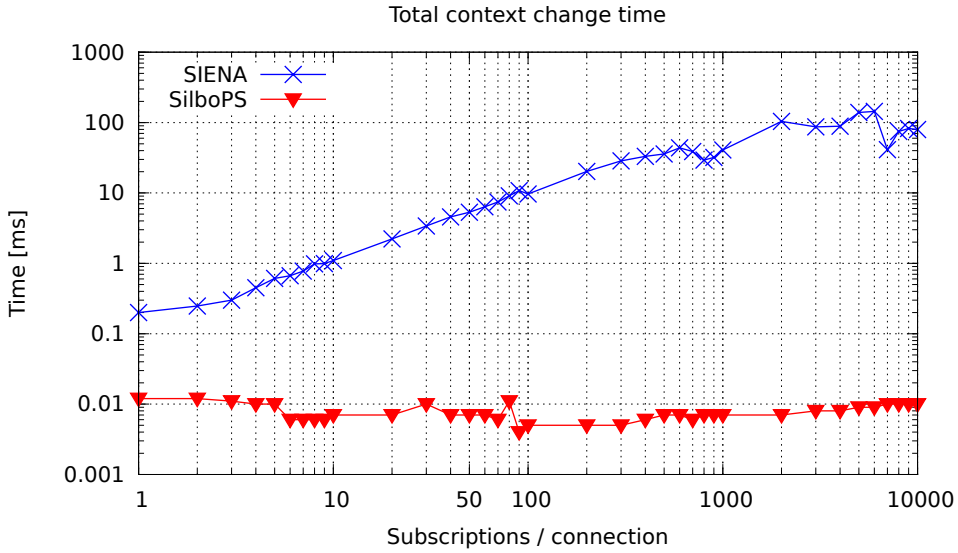


Figure 10 reports the time to process context change messages in a network of five brokers each with one publisher and 100 subscribers, where each subscriber sends on average 10 subscriptions for a total of 1,000 subscriptions per broker.

Our implementation is clearly about five times faster than SIENA. However, network transport protocol overhead reduces the three-order-of-magnitude gain shown earlier. Another network effect is buffering introduced by sockets. Note that the slope of the two curves changes at around 200 and 1000 context changes for SIENA and our prototype respectively; this is due to selected channel socket buffer filling. As a matter of fact, both systems use a queue to hold and then a single thread to process incoming messages in the broker. Consequently, incoming sockets do not slow down the routing since they each have their own thread. However, outgoing sockets do slow down the process because they use the broker's thread.

Figure 10. Time required for changing the context in a 5-broker network with 1 publisher and 100 subscribers each.

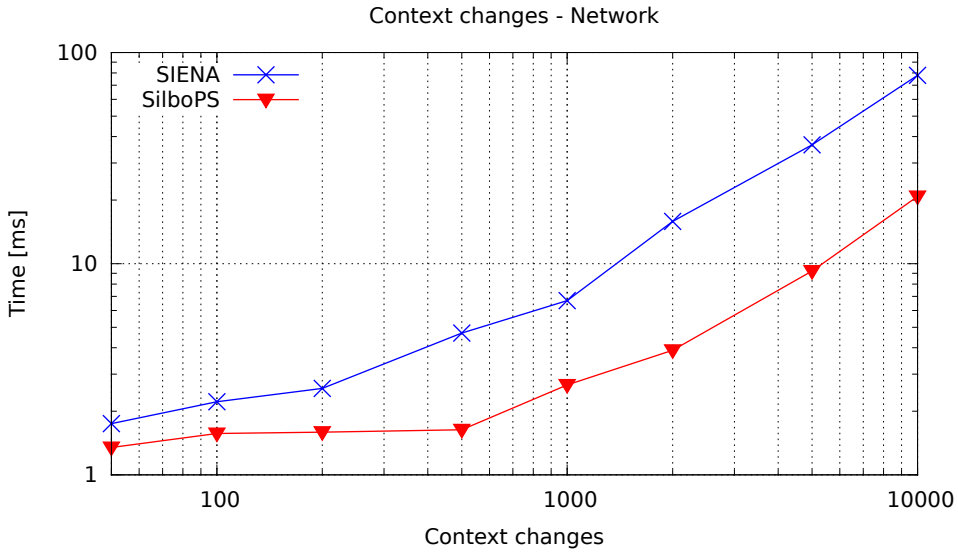


Figure 11 shows notification throughput when the broker is loaded with 10,000 subscriptions in the top chart, and with 80,000 subscriptions in the bottom chart. It is clear that SilboPS introduces a small overhead with respect to SIENA and that maintains the same shape through all the range; this proves our first assumption, stating that filters applied to notifications are stricter than the ones applied to context. As expected, the more notifications we send, the more stable the value is, since the broker queue always has an element and thus the broker saturation point can be reached.

These data are useful for calculating the matching time difference between the two systems. This reveals the trade-off point between the number of notifications and the number of changing subscriptions in the system, namely when it is convenient to switch from SIENA to our approach. We chose these values (see Table 1) because they represent the biggest difference between the two system's throughputs above the broker saturation point.

Figure 11. Notification throughput with 10,000 and 80,000 subscriptions in a single broker.

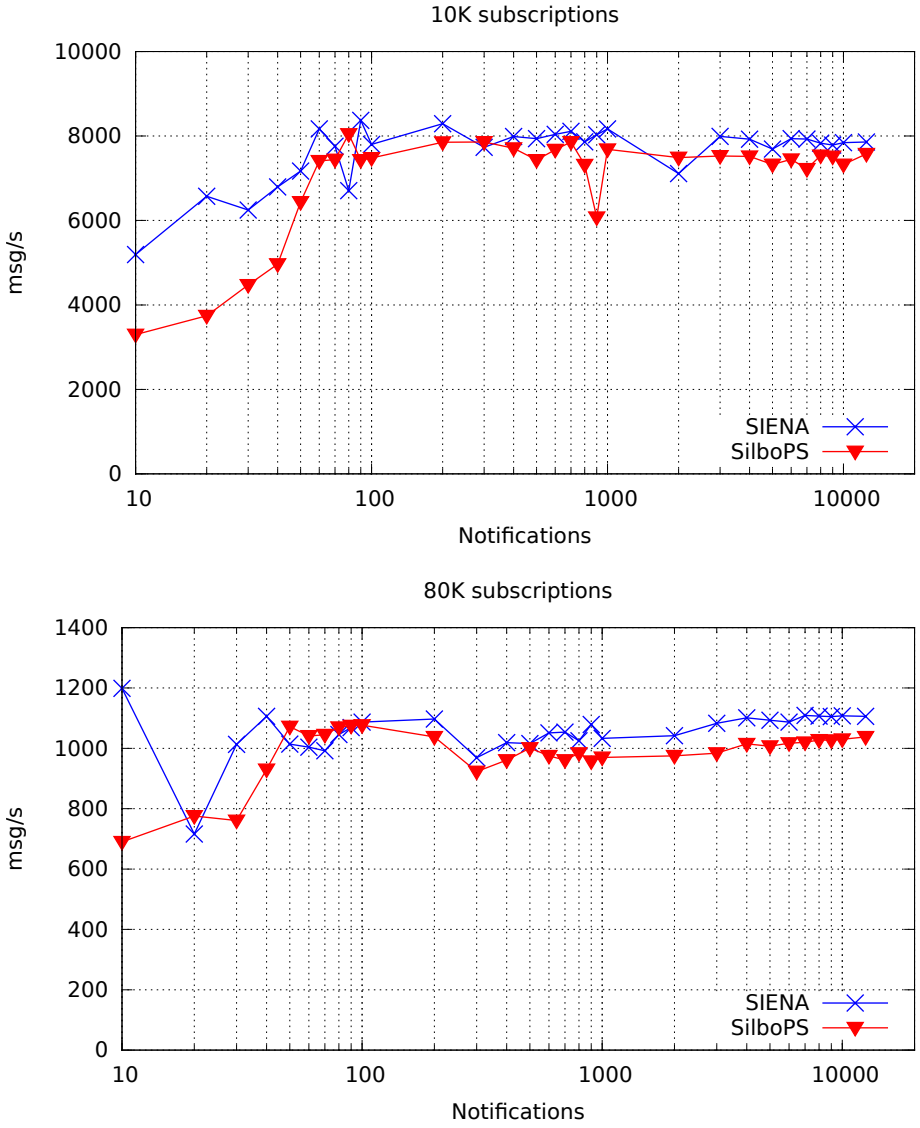


Table 1. Numerical result of test with 80,000 subscriptions.

Notifications	**SIENA**		**Explicit Context**		**Difference**	
	[msg/s]	**[us/msg]**	**[msg/s]**	**[us/msg]**	**[msg/s]**	**[us/msg]**
3,000	1,083	923	984	1,016	99	93
4,000	1,101	908	1,016	984	85	76
Average	1,092	915	1,000	1,000	92	85

The trade-off point can be found by comparing the matching and reconfiguration time for the two systems. More formally:

$X_{notification}$ is the number of notifications

$X_{context}$ is the number of context changes

T^{not}_{SIENA} is the average time to process a notification with SIENA, namely 915 µs/notification

T^{cxt}_{SIENA} is the average time to process a context change with SIENA, namely 105 µs/context change

$T^{not}_{SilboPS}$ is the average time to process a notification with SilboPS, namely 1,000 µs/notification

$T^{cxt}_{SilboPS}$ is the average time to process a context change with SilboPS, namely 0.2 µs/context change

Thus the inequality to be applied to determine when switch to SIENA is

$$X_{notification}T^{not}_{SIENA} + X_{context}T^{cxt}_{SIENA} < X_{notification}T^{not}_{SilboPS} + X_{context}T^{cxt}_{SilboPS} \quad (1)$$

which is solved with respect to the ratio of X's:

$$\frac{X_{notification}}{X_{context}} > \frac{T^{cxt}_{SilboPS} - T^{cxt}_{SIENA}}{T^{not}_{SIENA} - T^{not}_{SilboPS}} \quad (2)$$

and substituting the above values, we found as trade-off point:

$$\frac{X_{notification}}{X_{context}} \approx 1.23$$

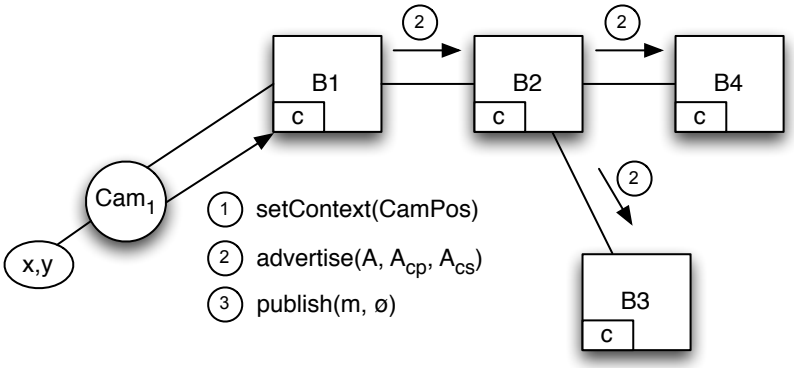
Consequently, if the ratio between $X_{notification}$ and $X_{context}$ is greater than 1.23, the SIENA processing time is better, whereas SilboPS performs better when the number of context changes is greater than the number of notifications. Note, additionally, that $X_{notification}$ and $X_{context}$ represent the *total* number, so the more subscribers in the system there are, the more likely our system is to perform better. Moreover, this supports our assumption for the scenario described in this paper, where the number of notifications (traffic issues) is lower than the number of context changes (car's position updates).

7. Future Work

Of the design principles presented in Section 4, we will focus now on the *generalized P/S model for contextual scoping.* According to this model, symmetric context-scoping filters are attached to both subscriptions and publications. This allows publishers to restrict notification delivery to subscribers (*i.e.*, by checking subscriber filter validity) based on both contexts. This scoping mechanism can be seen as role-based access control by the publisher [26,30] and as a new filtering dimension by the subscriber.

This design principle will be accomplished in two stages. First, a third parameter A_{cs} (*i.e.*, $advertise(A, A_{cp}, A_{cs})$ will be added to the advertisement $advertise(A, A_{cp})$ sent by the publisher (Cam_1), see Figure 12), where A_{cs} models the context information that the publisher is going to exploit to restrict the dissemination of published messages, resembling the above role-based access control.

Figure 12. Adding a new publisher to the scenario.



This advertisement is propagated through the broker network to B_2, B_3 and B_4, leading to changes in all their advertisement tables, which now manage the publisher advertisement context for subscribers, A_{cs}:

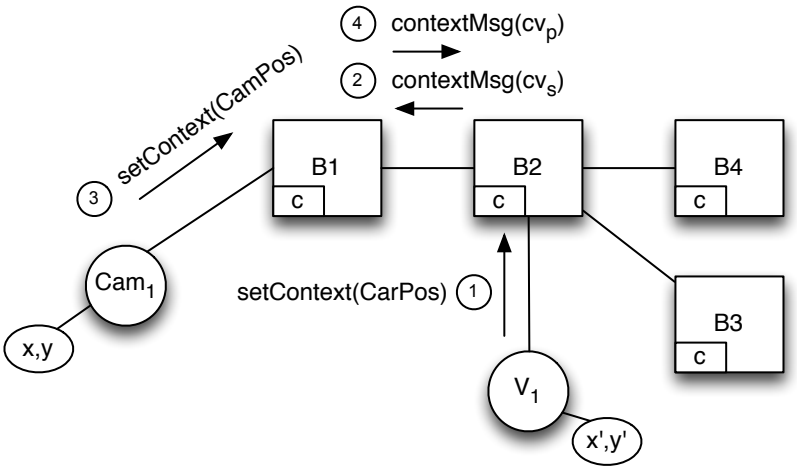
- B_1, *advertisement* table: $\langle Cam_1; A, A_{cp}, A_{cs}; \varnothing \rangle$
- B_2, *advertisement* table: $\langle B_1; A, A_{cp}, A_{cs}; \varnothing \rangle$
- B_3, *advertisement* table: $\langle B_2; A, A_{cp}, A_{cs}; \varnothing \rangle$
- B_4, *advertisement* table: $\langle B_2; A, A_{cp}, A_{cs}; \varnothing \rangle$

The second stage is concerned with enhancing the *publish* message, changing context values with a contextual scope filter F_{cs} (*i.e.*, a function of the subscriber context) that publishers (*i.e.*, cameras) use to restrict the propagation of m, and second context values in a similar way as for subscribers with *setContext*(*CamPos*) message. As Cam_1 does not restrict the scope, $F_{cs} = \varnothing$.

This way, Cam_1 can send notifications to the P/S infrastructure. B_1 receives $publish(m, F_{cs})$ where m is the notification, which is context-free content, and F_{cs} is a contextual scope filter that the camera uses to restrict the propagation of m.

If the traffic camera is a mobile sensor and changes position, it will send new *setContext* messages to B_1. The context manager of this broker will just update its local variables x_p and y_p with the new data and forward the message to its neighbors (see Figure 13). Further evaluation is needed regarding whether filter removal should be applied at each context change, either timed or advertisement-based.

Figure 13. Clients change their context.



The next steps are to evaluate the overall system performance overhead introduced by this approach, and the trade-off between this overhead and benefits for publishers that intend to take advantage of the added functionalities.

8. Conclusions

In this article, we have presented a novel context-aware P/S model in which the context is managed explicitly. Our model adds a new dimension to CBPS flexibility: context scoping based on context-invariant filters, *i.e.*, the notification filtering is separated from context filtering, allowing

SIENA-like forwarding index to be updated only on new subscriptions and at the same time enabling a quick subscriber's context update without rebuilding the whole structure, thanks to the indirection provided by our context functions and the *contextManager* table.

Our experiments have shown that the approach presented in this article introduces a small overhead with respect to SIENA in terms of notifications throughput, but at the same time outperforms SIENA by three orders of magnitude on subscriber context updates. In addition, using our solution, clients do not have to hold the whole list of sent filters and manage filter life cycle (unsubscribe previous subscriptions, create filters with the new context and send the new subscriptions).

We have also demonstrated that our solution is extremely valuable in domains where context is highly dynamic and the number of context changes exceeds the number of notifications, such as inventorying, stock portfolios, people or vehicle locators that have with high or "bursty" context update rates.

Finally, note that the reported research is being developed and applied as part of the 4CaaSt platform-as-a-service (PaaS) project (http://4caast.morfeo-project.org), where P/S is offered as both a value-added service to hosted applications and a key internal platform asset.

Acknowledgements

The research leading to these results was partially funded by the 4CaaSt project (http://www.4caast.eu/) from the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement No 258862. This article expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

1. Freris, N.M.; Kowshik, H.; Kumar, P.R. Fundamentals of large sensor networks: Connectivity, capacity, clocks and computation. *IEEE Proc.* **2010**, *98*, 1828–1846.
2. Baldauf, M.; Dustdar, S.; Rosenberg, F. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquit. Comput.* **2007**, *2*, 263–277.
3. Cugola, G.; Migliavacca, M. A context and content-based routing protocol for mobile sensor networks. *Lect. Note. Comput. Sci.* **2009**, *5432*, 69–85.
4. Geiger, L.; Durr, F.; Rothermel, K. On Contextcast: A Context-Aware Communication Mechanism. In Proceedings of the IEEE International Conference on Communications, Dresden, Germany, 14–18 June 2009; pp. 1–5.
5. Bader, S.; Nyolt, M. A Context-Aware Publish-Subscribe Middleware for Distributed Smart Environments. In Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, Lugano, Switzerland, 19–23 March 2012; pp. 100–104.
6. Cugola, G.; De Cote, J. On Introducing Location Awareness in Publish-subscribe Middleware. In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops, Columbus, OH, USA, 6–10 June 2005; pp. 377–382.

7. Cugola, G.; Margara, A.; Migliavacca, M. Context-Aware Publish-Subscribe: Model, Implementation, and Evaluation. In Proceedings of the IEEE Symposium on Computers and Communications, Sousse, Tunisia, 5–8 July 2009; pp. 875–881.
8. Bainomugisha, E.; Paridel, K.; Vallejos, J.; Berbers, Y.; Meuter, W. Flexub: Dynamic subscriptions for publish/subscribe systems in MANETs. *Lect. Notes Comput. Sci.* **2012**, *7272*, 132–139.
9. Musolesi, M.; Mascolo, C. CAR: Context-aware adaptive routing for delay-tolerant mobile networks. *IEEE Trans. Mob. Comput.* **2009**, *8*, 246–260.
10. Yasar, A.U.H.; Preuveneers, D.; Berbers, Y. Evaluation framework for adaptive context-aware routing in large scale mobile peer-to-peer systems. *Peer-to-Peer Netw. Appl.* **2011**, *4*, 37–49.
11. Carzaniga, A.; Rosenblum, D.S.; Wolf, A.L. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.* **2001**, *19*, 332–383.
12. Carzaniga, A.; Wolf, A.L. Forwarding in A Content-Based Network. In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Karlsruhe, Germany, 25–29 August 2003; pp. 163–174.
13. Cugola, G.; Margara, A. High-performance location-aware publish-subscribe on GPUs. *Lect. Notes Comput. Sci.* **2012**, *7662*, 312–331.
14. Sadoghi, M.; Jacobsen, H.A. BE-Tree: An Index Structure to Efficiently Match Boolean Expressions Over High-Dimensional Discrete Space. In Proceedings of the 2011 International Conference on Management of Data, Athens, Greece, 12–16 June 2011; pp. 637–648.
15. Chen, G.; Kotz, D. *A Survey of Context-Aware Mobile Computing Research*; Dartmouth College: Hanover, NH, USA, 2000.
16. Hong, J.-Y.; Suh, E.-H.; Kim, S.-J. Context-aware systems: A literature review and classification. *Exp. Syst. Appl.* **2009**, *36*, 8509–8522.
17. Kapitsaki, G.M.; Prezerakos, G.N.; Tselikas, N.D.; Venieris, I.S. Context-aware service engineering: A survey. *J. Syst. Softw.* **2009**, *82*, 1285–1297.
18. Bellavista, P.; Corradi, A.; Fanelli, M.; Foschini, L. A survey of context data distribution for mobile ubiquitous systems. *ACM Comput. Surv.* **2012**, doi:10.1145/2333112.2333119.
19. Benou, P.; Vassilakis, C. A context management architecture for m-commerce applications. *Cent. Eur. J. Comput. Sci.* **2012**, *2*, 87–117.
20. Dittrich, J.P.; Fischer, P.M.; Kossmann, D. AGILE: Adaptative Indexing for Context-Aware Information Filters. In Proceedings of the 24th International Conference on Management of Data, Baltimore, MD, USA, 14–16 June 2005.
21. Burcea, I.; Jacobsen, H.A. L-ToPSS—Push-oriented location-based services. *Lect. Notes Comput. Sci.* **2003**, *2819*, 131–142.
22. Xu, Z.; Jacobsen, H.A. Efficient Constraint Processing for Highly Personalized Location Based Services. In Proceedings of the Thirtieth International Conference on very Large Data Bases, Toronto, ON, Canada, 29 August–3 September 2004; pp. 1285–1288.
23. Chen, X.; Chen, Y.; Rao, F. An Efficient Spatial P/S System for Intelligent Location-Based Services. In Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, San Diego, CA, USA, 8 June 2003; pp. 1–6.

24. Fu, K.K. Mobile Spatial Subscriptions for Location-Aware Services. M.Sc. Thesis, University of Waterloo, Waterloo, ON, Canada, 2010.
25. Dey, A.K. Understanding and using context. *Pers. Ubiquit. Comput.* **2001**, *5*, 4–7.
26. Bacon, J.; Eyers, D.M.; Singh, J.; Pietzuch, P.R. Access Control in Publish/subscribe Systems. In Proceedings of the 2nd International Conference on Distributed Event-based Systems, Rome, Italy, 2–4 July 2008; pp. 23–34.
27. Li, G.; Muthusamy, V.; Jacobsen, H.A. Adaptive content-based routing in general overlay topologies. *Lect. Note. Comput. Sci.* **2008**, *5346*, 1–21.
28. Kazemzadeh, R.S.; Jacobsen, H.A. Partition-Tolerant Distributed Publish/Subscribe Systems. In Proceedings of the 2011 IEEE 30th International Symposium on Reliable Distributed Systems, Madrid, Spain, 4–7 October 2011; pp. 101–110.
29. Oracle javadoc 7. Java™ Platform, Standard Edition 7 API Specification. http://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime() (accessed on 28 February 2013).
30. Belokosztolszki, A.; Eyers, D.M.; Pietzuch, P.R.; Bacon, J.; Moody, K. Role-Based Access Control for Publish/subscribe Middleware Architectures. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems*, San Diego, CA, USA, 8 June 2003; pp. 1–8.