On Context-Aware Publish-Subscribe

Gianpaolo Cugola
Dip. Elettronica e Informazione
Politecnico di Milano, Italy
cugola@elet.polimi.it

Matteo Migliavacca
Dip. Elettronica e Informazione
Politecnico di Milano, Italy
migliava@elet.polimi.it

ABSTRACT

Complex communication patterns often need to take into account the characteristics of the environment, or the situation, in which the information to be communicated is produced or consumed. Publish-subscribe, and particularly its content-based incarnation, is often used to convey this information by encoding the "context" of the publisher into the published messages, taking advantage of the expressiveness of content-based addressing to encode context-aware communication patterns. In this paper we claim that this approach is both inadequate and inefficient and propose a context-aware publish-subscribe model of communication as a better alternative. In particular, we describe the API of a new publish-subscribe model that is both content and context-based, and we explore possible routing schemas to implement this new model in a distributed publish-subscribe system potentially improving traditional content-based routing.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems

Keywords

Publish-subscribe, Context, Content-Based Routing

1. MOTIVATION

Content-based publish-subscribe [2] provides the ability of addressing messages based on their content, resulting in a strong decoupling among communicating parties. On the other hand, an effective communication paradigm must take into account the characteristics of the environment, or the situation, in which the information to be communicated is produced or consumed, i.e., the communicating parties' *context*. As an example, consider a fire monitoring system deployed in a large building. Each room is equipped with a communication infrastructure connecting multiple sensors and actuators. Sprinklers subscribe for alert messages generated within a short distance from their position while detectors publish alerts when they detect smoke (Table 1).

sprinklers	**subscribe**($msgType = Alert \wedge$ $myPos.x - 50 \leq x \leq myPos.x + 50 \wedge$ $myPos.y - 50 \leq y \leq myPos.y + 50$)
smoke detectors	**publish**($msgType = Alert$, $x = myPos.x$, $y = myPos.y$)
signals	**subscribe**($msgType = Evacuate* \wedge$ $x = myPos.x \wedge y = myPos.y$)
planner	**subscribe**($msgType = Alert$) **publish**($msgType = EvacuateL$, $x < fire.x$) **publish**($msgType = EvacuateR$, $x > fire.x$)

Table 1: Fire Monitoring System (FMS)

Even this simple example shows how the need of *context-awareness* is so common in publish-subscribe applications that the publisher's context is usually encoded into messages, in a tentative to take advantage of the expressiveness of content-based addressing to encode context-aware communication patterns. We claim that this approach is both inadequate and inefficient, especially for large scale systems featuring a distributed dispatcher, thus motivating the adoption of a new publish subscribe model of communication that takes the context of both publishers and subscribers as an additional, explicit parameter.

Inversion of Matching. Suppose illuminated signals are deployed in corridors to direct people in case of evacuation. An evacuation planning system, according to the alerts received, computes an evacuation plan and publishes a message toward the signals at the east of the detected fire to make them display an eastbound arrow, towards an eastern emergency exit. Similarly, another message is published toward the signals located in the area at the west of the fire to do the opposite, directing people to the western exit. To correctly route these messages, the subscribers (i.e., the signals) should specify their location into subscriptions, while the publisher (i.e., the planner) should add a constraint into the messages it publishes to reach the required signals, only (see the last two rows of Table 1). Unfortunately, this is not possible in conventional content-based publish-subscribe systems where messages are designed to hold single values for each attribute, while subscriptions contains constraints on these values. This means that both the data model and the corresponding matching semantics of traditional content-based publish-subscribe systems are unsuited for the case under consideration, which should be "reversed" to cope with the case we consider[1].

[1]The few content-based publish-subscribe systems that do

DEBS '08, June 2–4, 2008 Rome, Italy

sprinklers	**subscribe**($msgType = Alert$; $myPos.x - 50 \leq x \leq myPos.x + 50 \wedge myPos.y - 50 \leq y \leq myPos.y + 50$)
smoke detectors	**setContext**($\{x = myPos.x, y = myPos.y\}$), **publish**($\{msgType = Alert, x = myPos.x, y = myPos.y\}$; ALL)
signals	**setContext**($\{x = myPos.x, y = myPos.y\}$), **subscribe**($msgType = Evacuate*$; ALL)
planner	**subscribe**($msgType = Alert$; ALL) **publish**($\{msgType = EvacuateL\}$; $x < fire.x$) **publish**($\{msgType = EvacuateR\}$; $x > fire.x$)

Table 2: FMS using Context-Aware API

Efficiency. The second reason to explicitly introduce context into publish-subscribe is efficiency in distributed dispatching scenarios. Dealing explicitly with context allows to limit the spreading of subscriptions only to those areas of the routing network where matching publishers exist (i.e., those whose context satisfies the context filter specified by the subscriber). This reduces the overhead of the subscription and unsubscription processes (saving bandwidth), while also reducing the time required to match messages, since the routing tables are smaller.

2. API AND ROUTING

To overcome the limitations above, we propose to introduce context-awareness into the publish-subscribe API. Each client $n \in \mathcal{N}$ can set its current context $c \in \mathcal{C}$ by invoking a special **setContext**(c) operation. Additionally, n can publish messages $m \in \mathcal{M}$ for subscribers whose context matches the context filter f_{ctx} by invoking the operation **publish**($m; f_{ctx}$). Similarly, n can subscribe to messages matching the content filter f_{msg} and coming from publishers whose context matches the context filter f_{ctx} through the operation **subscribe**($f_{msg}; f_{ctx}$), while the operation **unsubscribe**($f_{msg}; f_{ctx}$) does the opposite. Finally, when the client n receives a message m, the publish-subscribe service invokes a **receive**(m) callback. Table 2 shows how the Fire Monitoring System example can be easily implemented with these context-aware publish-subscribe primitives.

Routing Approaches. In context-aware publish-subscribe message delivery is determined by three factors: the context of the publisher c_p, the context of the subscriber c_s, and the content of the message m. Each of these elements must match a corresponding filter: the content filter f_{msg_s} and the context filter f_{ctx_s} set by the subscriber, plus the context filter f_{ctx_p} set by the publisher. Starting from the observation that this match can be completed only when the message is published, we might devise three routing approaches. The first one is to diffuse (c_p, m, f_{ctx_p}) from the publisher toward the potential subscribers when the message m is published (**publisher forwarding**), filtering these publications when they reach the subscriber's broker[2]. However, without any knowledge about which brokers host possible matching subscriptions, such publications must flood the network to reach the subscribers' brokers, where matching happens. This situation can be improved by diffusing $(c_s, f_{msg_s}, f_{ctx_s})$ information from the subscriber when a subscription is issued (**subscriber forwarding**), thus establishing routes to be followed back by messages. As in the previous case, since the brokers hosting clients whose context matches the context filter f_{ctx_s} are not known a priori, such subscriptions must flood the network. This can be avoided by diffusing contextual information before any subscription is issued: each broker forwards the context of the attached clients in the network, thus establishing "context" routes that will be used to forward both subscriptions and messages. Subscriptions are steered accordingly reaching only those brokers holding clients with matching contexts, thus establishing "subscription" routes to be followed back by messages. This approach (**context forwarding**) has two advantages over the previous ones: it reduces the overhead of subscribing (since subscriptions are routed only toward clients with matching contexts) and reduces the time required to match messages, since the context of the publisher has been already (i.e., at subscription time) been matched against the context filter of the subscriber. Of course, if the context of clients changes frequently, this approach, which require this information to be flooded, may become inefficient.

The description above, even if very abstract, allow us to make some initial comparisons about the three routing approaches. In particular, publisher forwarding incurs a lot of overhead to route messages, on the other hand it does not forward around any other information. As a consequence, it provides the best performance in situations in which subscriptions and contexts change very frequently. The second approach requires subscriptions to be flooded, so it is suited to situations in which contexts and interests of clients do not change frequently, at least w.r.t. the frequency of publications. The third approach is perfect when contexts are fairly static. It minimizes the cost of subscribing and also that of publishing, at the price of an overhead to keep context information up to date.

3. CONCLUSION

In most of the scenarios in which publish-subscribe is used, the context, being that of the publisher or that of the subscriber, would be a useful information, if available, to limit the scope of communication.

Content-based publish-subscribe is a very expressive model of communication but it cannot entirely capture truly context-aware communications patterns. To overcome this limitation, we proposed a context-aware extension to the publish-subscribe model of communication. Our short term goal is to show how it can be efficiently implemented in a distributed publish-subscribe system, and to test to which degree it can outperform traditional content-based routing protocols in medium to large scale scenarios whenever contextual information is used by publishers and subscribers.

4. REFERENCES

[1] G. Mühl and L. Fiege. Supporting covering and merging in content-based publish/subscribe systems: Beyond name/value pairs. *IEEE Distributed Systems Online (DSOnline)*, 2(7), 2001.

[2] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer, 2006.

not suffer of this problem are those adopting a Turing-complete language to implement filters, which however are hard to optimize [1]

[2]We suppose that each broker stores the relevant information (i.e., contexts and filters) of the clients it serves.