Design and Evaluation of a Publish/Subscribe Framework for Ubiquitous Systems

Zigor Salvador, Alberto Lafuente, and Mikel Larrea

University of the Basque Country UPV/EHU
Donostia-San Sebastián, Spain
{zigor.salvador,alberto.lafuente,mikel.larrea}@ehu.es

Abstract. This paper describes the design and evaluation of a novel publish/subscribe communication framework for ubiquitous systems and applications. The motivation of this work is the realization of the fact that the publish/subscribe communication model has several features that make it suitable to serve as a communication substrate for ubiquitous systems. In particular, we argue that a publish/subscribe framework that is scalable and supports client mobility is a valuable asset for the development of ubiquitous applications. We present a reference implementation, Phoenix, that supports the deployment of publish/subscribe components in mobile devices such as smartphones. In addition, we evaluate the functionality of Phoenix and its performance, in order to determine its operational constraints for server and mobile platforms.

Keywords: publish/subscribe, client mobility, ubiquitous environments, software implementation, empirical validation, performance evaluation.

1 Introduction

The client/server model has played a foundational role in the ongoing success of distributed systems and the Internet. However, distributed systems that are based on the traditional client/server model exhibit a tight coupling of components. Consequently, the deployment and maintenance of large-scale distributed systems based on the client/server model is a complex task. This becomes even more evident when the distributed systems are composed of heterogeneous services, devices and flows of information, as is the case with ubiquitous systems.

The publish/subscribe *communication model* or *interaction paradigm* overcomes this limitation by introducing an indirection layer that decouples components, i.e., producers and consumers of information [21]. This enables the creation of flexible and robust distributed systems that exploit the benefits of space, time and synchronization decoupling of components. The decoupling of components increases scalability by removing all explicit dependencies between the interacting participants [7]. Removing these dependencies reduces coordination and synchronization requirements and makes the publish/subscribe communication model suitable for distributed environments that are asynchronous by nature, such as ubiquitous systems that seek to integrate mobile devices [10].

K. Zheng, M. Li, and H. Jiang (Eds.): MOBIQUITOUS 2012, LNICST 120, pp. 50–63, 2013.

In publish/subscribe *systems*, information producers are known as *publishers* and information consumers are referred to as *subscribers*. Publishers generate information in the form of *events* while an independent Event Notification Service handles the task of delivering those events to a subset of the subscribers. The use of an explicit layer of indirection provides an overall separation of concerns and is what makes publish/subscribe systems inherently flexible and scalable.

Publish/subscribe *applications* are the result of using a publish/subscribe communication model to orchestrate distributed application components. In essence, publish/subscribe applications employ publishers and subscribers to implement a series of information flows or *content streams* [30]. Content streams are continuous flows of information that are transported from a publisher component to subscriber components by the publish/subscribe infrastructure. In order to shape the content streams of a publish/subscribe system, the use of content-based filtering mechanisms can be requested by subscriber components.

Publish/subscribe applications are characterized by the number of client components they employ and the content streams or events they exchange. Assuming the availability of a broker infrastructure, a minimal application can be created with a publisher, a subscriber and a single content stream. However, real applications exhibit a higher cardinality with regards to both the number of client components and the volume of content streams they employ. Depending on the application domain, the amount of publish/subscribe components can range from a handful to tens of thousands. This is often the case with ubiquitous systems and their applications, where having a large number of components is expected.

In general, the decoupled and data-centric nature of the publish/subscribe interaction paradigm makes it a suitable foundation for a wide variety of applications. In particular, the most suitable applications for this communication model are the ones that require the timely, efficient and scalable dissemination of information between large numbers of components [16]. Note that large-scale systems can result either from the geographic distribution of nodes or from the concentration of nodes on a relatively small area [23]. Examples of publish/subscribe application domains include network monitoring, financial services, mobile computing, industrial automation, social networks, smart environments, scientific computation, content distribution and sensor networks [13] [23] [29].

Taking into account that the software architecture of a ubiquitous system has to overcome a series of specific issues and challenges [6], we argue that the underlying communication infrastructure of a ubiquitous system should provide:

- Heterogeneity: allowing a variety of devices and services to operate.
- Dependability: avoiding severe failures that happen frequently.
- Scalability: enabling the deployment of large-scale systems.
- Mobility: enabling the users to roam the environment.

The publish/subscribe communication model not only provides these features but also supports relevant concepts such as *localised scalability* [27] and *content-centric networking* [11]. As a consequence, we conclude that ubiquitous systems and applications can benefit from the use of *content-based* publish/subscribe [7].

The present work documents a foray into this idea, with the objective of validating the applications of our previous work in the field [25]. To that end, we introduce *Phoenix*, a novel publish/subscribe communication framework for ubiquitous systems, and illustrate the empirical evaluation process and first-hand experimentation that has been carried out in order to validate our approach.

The remainder of this paper is structured as follows: in Section 2 we describe the context and features of our publish/subscribe implementation. In Section 3 we present the process that has been carried out to validate the implementation with a ubiquitous application that includes mobile clients. In Section 4 we introduce a timeliness model for our framework and measure the performance of the system. Finally, Section 5 outlines our findings and concludes the paper.

2 The Phoenix Publish/Subscribe Framework

In this section we describe our approach to publish/subscribe and client mobility. We also cover the reference implementation, Phoenix, and key related work.

2.1 Service Interface

Publish/subscribe is a communication model where information producers (publishers) and information consumers (subscribers) exchange information (events) by means of an Event Notification Service that is composed of a set of *brokers*. Publishers publish events and subscribers make use of predicate-based filters to subscribe to specific kinds of events. The distributed network of brokers *matches* published events and subscriptions, routing them as required. Figure 1 shows the interface provided to the clients of a content-based publish/subscribe system.

```
public void publish(Event notification);
public void subscribe(Filter subscription);
public void unsubscribe(Filter subscription);
```

Fig. 1. Interface for publish/subscribe applications

2.2 Client Mobility

Publish/subscribe clients have a *reference broker* that mediates between them and the rest of the system. When a client is hosted in a mobile device, sudden disconnections can take place as a result of limited wireless coverage. As a result, the link between a client and its reference broker will break and publish/subscribe service will be disrupted. Upon the recovery of wireless connectivity, clients may connect to the same reference broker and resume their operation. However, the physical mobility of a device may prevent this and a client could be required to

migrate to a new reference broker. In order to cope with these situations, the Event Notification Service needs to handle client mobility and disconnections.

To that end, we propose the use of a publish/subscribe routing algorithm that warrants the transparent reconnection of clients [25]. Based on a Simple Routing strategy [1] our custom routing algorithm ensures that the routing tables of brokers are updated to reflect the mobility of their clients. In addition, our routing algorithm can optionally replay published events that may be pending delivery as a result of a disconnection. A detailed description of the contributed routing algorithm and its correctness proof can be found in [26]. Figure 2 shows the mobility related interface provided to subscribers. Note that publisher mobility is transparent to the system as a result of extending a Simple Routing strategy.

```
public void migrate(boolean replay);
public void resume(boolean replay);
```

Fig. 2. Interface for migration and resumption operations

2.3 Design and Implementation

In order to evaluate our approach, we have implemented a reference content-based publish/subscribe framework known as Phoenix [24]. The main objective of Phoenix is to provide a robust and scalable communication infrastructure for ubiquitous systems where mobility plays a key role. Therefore, it takes into account the requirements that have already been outlined and implements a custom routing algorithm that supports communication efficient client mobility.

Phoenix has been implemented in Java in order to enable the use of publish/subscribe application components in a wide variety of devices and platforms. In particular, Phoenix supports the deployment of publishers and subscribers in Android devices. Phoenix makes use of the Apache MINA network application framework to provide asynchronous communication capabilities to both the publish/subscribe components and, by extension, the high-level services and applications that are built on top of them. Internally, the publish/subscribe components of the framework communicate by means of JSON formatted message passing over TCP/IP streams. Other noteworthy features of Phoenix include:

- Optimal routing of events and communication efficient client hand-offs.
- Automatic discovery of brokers by means of multicast and unicast.
- Interoperable message serialization/deserialization mechanism.
- Graphical user interface for the management of brokers.

The middleware layer of the Phoenix framework incorporates over 7,000 lines of code and provides developers with simple interfaces such as the ones depicted in Figures 1 and 2. Note that Phoenix has been released[1] under the MIT license.

[1] The Phoenix source code repository is `https://github.com/zigorsalvador/phoenix`

Indeed, a key motivation behind the development of Phoenix stems from the fact that although several academic contributions have been published in the field, few open source implementations are available and none of them provides key features such as client mobility support and/or full integration with Android.

2.4 Related Work

JEDI was the first publish/subscribe system to support client mobility [5]. In particular, JEDI implements a client hand-off or migration protocol that requires clients to proactively inform the middleware before moving away from a broker. This initial *move out* operation must then be followed by a *move in* operation that triggers the reconfiguration of the system. As a consequence, the client mobility protocol of JEDI is not suitable for situations where clients suffer sudden disconnections from the infrastructure [29]. Additionally, it must be noted that in order to support migrations, JEDI requires the reissuing of subscriptions and the availability of out-of-band communication between the pair of brokers involved in the migration. Furthermore, migrations in JEDI often result in the delivery of *duplicate events* due to the coexistence of old and new delivery routes.

The Siena publish/subscribe system was also among the first to support client mobility [3]. In particular, a generic Mobility Service was implemented in order to validate a client hand-off protocol that could be used with any publish/subscribe system or implementation. The Mobility Service relies in client proxies and explicit *move in* and *move out* operations and due to its generic nature requires no changes to the application programming interface of the system. On the other hand, the use of the Mobility Service results in a high messaging overhead, due to the fact that it has to rely on message flooding to locate *source* and *destination* brokers. Indeed, the high signalling cost of this approach was modelled and found to be excessive [29], which severely limits its value for ubiquitous systems.

The REBECA publish/subscribe system was eventually extended to cope with client mobility [8] [9] [17] [32]. In particular, an algorithm for Roaming Clients was proposed in the context of a publish/subscribe system based on an acyclic graph topology with advertisement semantics. Note that REBECA does not support publisher mobility, due to the choice of advertisement semantics and the lack of a specific publisher mobility protocol. Moreover, the REBECA algorithm relies in the reissue of subscriptions upon migration, which is inefficient.

Client mobility support has also been an active research topic in the context of the PADRES publish/subscribe system. In particular, several contributions were targeted at publisher mobility and represent the first foray into this relevant aspect of client mobility support [18] [19] [20] [22]. Additionally, several performance evaluations were carried out regarding publish/subscribe scenarios where client mobility is generalized. One of the most interesting findings is that in most mobile scenarios the replay of buffered events dominates the signalling cost of subscriber hand-off protocols in real-world deployments of client mobility [2].

All in all, Phoenix builds upon previous contributions introduced by the aforementioned systems. However, the custom routing algorithm [25] employed by Phoenix has some advantages with respect to the preceding systems. For one,

Phoenix supports both publisher and subscriber mobility. For another, Phoenix implements a client hand-off protocol that is communication efficient, i.e., minimizes the amount of traffic generated as a result of client mobility. Furthermore, the formal correctness of our custom routing algorithm has been established and therefore Phoenix provides a solid foundation for incremental work in the field.

3 Validation

In this section we describe the empirical validation of the new communication framework with a combination of synthetic tests and application prototypes.

3.1 Synthetic Tests

Resumption and migration mechanisms have been validated using mobile devices. Figure 3 illustrates the Android application that has been developed to conduct the validation. The Mobility application is composed of a subscriber component and a user interface that enables researchers to request subscriptions, resumptions and migrations. Normally, the user launches the application and enables the wireless communication interface of the mobile device. Once the network interface is up, the user triggers the discovery of a broker in the local area network. If a broker has been found, the user can request the submission of a subscription to that broker, initiating the delivery of events. After an arbitrary amount of time, the user will disable the network interface, disrupting the communication channel between the subscriber and the broker. Then, the user will have to re-enable the network interface, and repeat the discovery procedure. When the identity of the broker that is discovered matches that of the broker that was available before the disconnection, the application will request a resumption. If, on the other hand, a new broker is discovered after the disconnection period, the application will issue a migration request. In either case, the broker will resume the delivery of new events and will replay any events that were not delivered during the disconnection. The Mobility application will check that all expected messages were successfully delivered. Figure 4 illustrates the traffic profiles of *source* and *destination* brokers during an example migration.

3.2 Functional Prototypes

The overall functionality and performance of the framework have been validated using realistic publish/subscribe application prototypes. Figure 5 shows a screenshot of a prototype developed for that purpose: the Tracker application. The idea behind the Tracker application is to implement a real-time visitor tracking system using the Phoenix communication framework. The web application is composed of a single subscriber component and a user interface that enables researchers to visually manage subscriptions and track the location of visitors on a map. In the figure, the red circles represent location-based subscriptions,

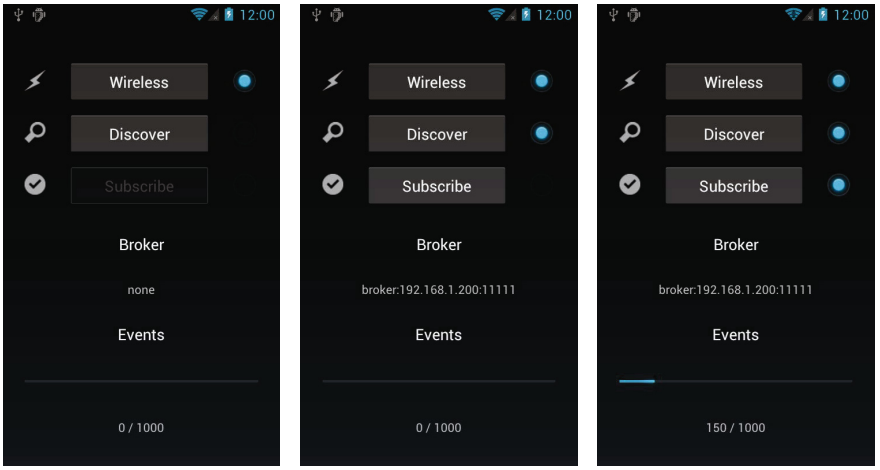

Fig. 3. Screenshot sequence of the Mobility application during a migration test

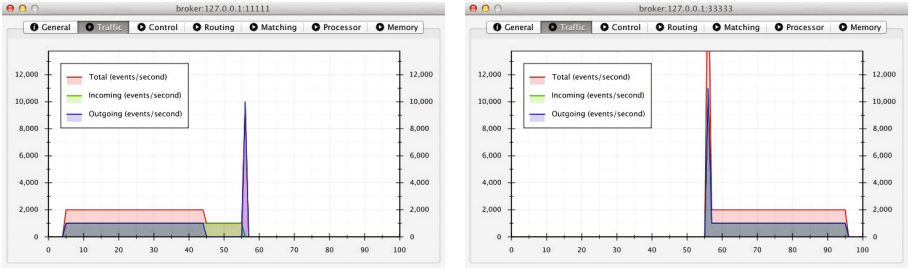

Fig. 4. Traffic profiles of source and destination brokers during a migration test

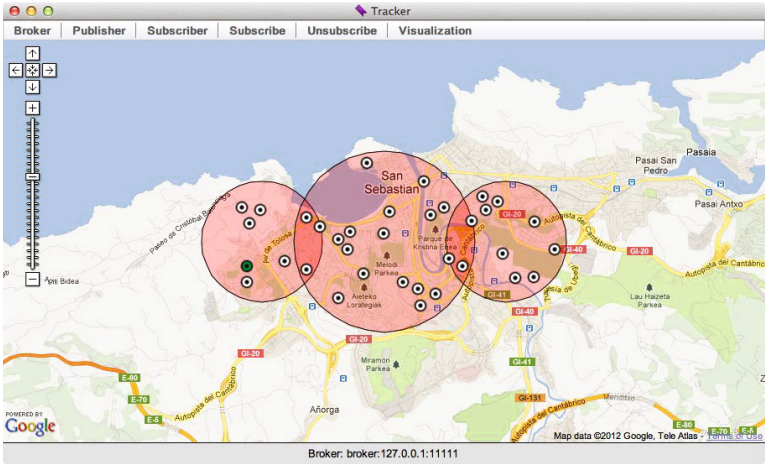

Fig. 5. Screenshot of a web browser rendering the Tracker application

while the small dots represent the real-time location of visitors. In order to generate location data, the mobile devices of visitors are equipped with publisher components that periodically publish the GPS location of these mobile devices and their owners. Using the web application, users can generate fine-grained location-based subscriptions which enable the system to filter incoming events and only deliver those that match a given subscription. The Tracker application has been tested with real subjects using Android devices. However, in order to validate the scalability of the application, virtual visitors can be created by the system and instructed to update their location with an arbitrary periodicity.

4 Performance

In this section we analyse the performance of Phoenix from three perspectives. First, we adopt a theoretical approach and perform an analysis of content-based publish/subscribe performance. Then, we present the results obtained in a series of experimental tests. Finally, we evaluate performance using Android devices.

4.1 Timeliness Model

We assume that the number of brokers is finite and their connection graph G is acyclic and static. Therefore, we can define the diameter of the broker graph as:

$$d = diameter(G)$$

We assume that the communication links in the system are timely, i.e., message transmission delays are bounded. As a result, we can model the communication times involved in the operation of the system and combine them with several assumptions regarding the processing times for different messages in order to model the performance or *average message dispatching times* of a our system. Let δ_c be the average time required for a message to traverse a link that connects a broker and one of its local clients and let δ_b be the average time required for a message to traverse a link that connects a broker and one of its neighbour brokers. If we assume that brokers will benefit from high capacity network links:

$$\delta_c \gg \delta_b$$

Once a given message reaches a process, the message has to be processed, involving changes to data structures and/or event matching calculations. Consequently, we can define the average processing time for each publish/subscribe primitive:

- σ_{SUB}: average time required for a broker to process a subscription message
- σ_{UNS}: average time required for a broker to process a unsubscription message
- σ_{PUB}: average time required for a broker to process a publication message

Based on the fact that the event matching problem is regarded as the main potential bottleneck of a publish/subscribe system [12] [13] [15], we assume that:

$$\sigma_{PUB} \gg \sigma_{SUB} \simeq \sigma_{UNS}$$

And based on the previous, we can model *average message dispatching times* for subscription, unsubscription and publication messages, respectively, as follows:

$$\Delta_{SUB} \simeq \delta_c + \sigma_{SUB} + d(\delta_b + \sigma_{SUB})$$

$$\Delta_{UNS} \simeq \delta_c + \sigma_{UNS} + d(\delta_b + \sigma_{UNS})$$

$$\Delta_{PUB} \leq \delta_c + \sigma_{PUB} + d(\delta_b + \sigma_{PUB}) + \delta_c$$

Finally, based on the assumptions that have been noted, we can now assert that:

$$\Delta_{PUB} \gg \Delta_{SUB} \simeq \Delta_{UNS}$$

This realization underlines the importance of the matching process in publish/subscribe systems. Furthermore, it motivates us to conduct a performance analysis that is focused in Δ_{PUB} scalability and the impact of different optimizations.

4.2 Empirical Evaluation

In order to measure the performance of Phoenix, we have deployed our publish/subscribe system in a dedicated cluster composed of eight server nodes running Ubuntu 11.10 with dual 2.4 GHz Xeon CPUs and 24 GBytes of RAM. Six of the nodes are exclusively dedicated to the execution of a broker overlay with a star topology. The two remaining nodes are dedicated to the execution of a *benchmarking application* and a *workload generator*. The benchmarking application is composed of a publisher and a subscriber component that exchange *probe messages* which traverse the broker overlay. Probe messages are time-stamped and enable the benchmarking application to measure two performance metrics: *probe latency* and *probe throughput*. The *workload generator* orchestrates the execution of synthetic clients and generates arbitrary amounts of background traffic. Probe messages and workload messages are, respectively, 286 and 886 bytes long.

In its basic, non-optimized form, the algorithm in [25] makes heavy use of event matching, which limits the scalability of the system. As a result, two optimizations have been implemented in Phoenix and considered in the experiments. The first optimization, *Filter Poset*, aims at reducing the *number of filters* that are involved in the matching operation of a given event. To do so, it exploits the relationship among the filters stored in the routing tables by maintaining a partially ordered set of filters [4] [31]. The second optimization, *Single Matching*, aims at reducing the *number of matching operations* that are involved in the delivery of a given event. To do so, it exploits global knowledge by having the front-end broker of each publisher compute the set of matching subscriptions on behalf of the rest of the brokers. This optimization is similar to the approach followed in [13] [14] [28]. The combination of the two optimizations results in four benchmarking configurations: $C1$ (no optimization), $C2$ (Single Matching), $C3$ (Filter Poset) and $C4$ (both optimizations). The workload consists of a set of publish/subscribe clients that generate various degrees of synthetic background traffic, in the form of spurious events that need to be dispatched. The performance factors that dimension workloads are: R_p: the rate of background publications in the

Table 1. Average probe throughput (events/second)

Workload	(R_p, N_s)	$C1$	$C2$	$C3$	$C4$
$W1$	$(0, 0)$	24,687	20,626	23,955	21,421
$W2$	$(0, 250)$	10,089	11,639	17,522	20,557
$W3$	$(0, 500)$	6,843	8,631	18,158	17,050
$W4$	$(5{,}000, 0)$	25,198	22,506	22,197	21,806
$W5$	$(5{,}000, 250)$	7,324	11,305	15,942	18,394
$W6$	$(5{,}000, 500)$	3,905	8,365	14,327	15,996
$W7$	$(10{,}000, 0)$	22,848	21,843	21,693	21,788
$W8$	$(10{,}000, 250)$	5,448	11,474	15,031	16,597
$W9$	$(10{,}000, 500)$	2,746	8,071	11,956	15,221

Table 2. Average probe latency (milliseconds)

Workload	(R_p, N_s)	$C1$	$C2$	$C3$	$C4$
$W1$	$(0, 0)$	1.002	1.118	1.079	1.137
$W2$	$(0, 250)$	1.247	1.109	1.060	1.175
$W3$	$(0, 500)$	1.451	1.168	1.030	1.144
$W4$	$(5{,}000, 0)$	1.063	1.079	1.022	1.229
$W5$	$(5{,}000, 250)$	1.827	1.462	1.583	1.634
$W6$	$(5{,}000, 500)$	2.873	1.630	1.764	1.573
$W7$	$(10{,}000, 0)$	1.048	1.092	1.063	1.047
$W8$	$(10{,}000, 250)$	3.121	1.749	1.926	1.556
$W9$	$(10{,}000, 500)$	38.891	1.740	2.108	1.649

Table 3. Average smartphone throughput (events/second) and latency (miliseconds)

Interface	Throughput	Latency
WIFI	148	11
UMTS	185	151

Table 4. Average smartphone battery life (minutes)

Scenario	Duration	Percentage
Baseline	582	100%
Publisher	252	43%
Subscriber	267	46%
Combination	215	37%

system and, N_s: the amount of global subscriptions in the system. Note that three discrete levels have been considered for each of the two cited performance factors. In particular, R_p can be one of 0, 1,000 or 2,000 events/second and N_s can be one of 0, 50 or 100 subscribers for each of the five border brokers. This translates into a global publication rate of 0, 5,000 or 10,000 events/second and a total of 0, 250 or 500 concurrent subscribers in the system. Note that two metrics, four configurations and nine workloads require 72 unique experiments.

Tables 1 and 2 show average probe throughputs and average probe latencies, respectively. Throughput measurements involve the publication of 50,000 events and were repeated 10 times. Latency measurements require a single event and were repeated 10, 000 times. Based on these results, we can highlight that the performance of the non-optimized configuration ($C1$) stalls under load ($W9$). However, in the highly-optimized configuration ($C4$), Phoenix manages to dispatch over 15,000 messages per second with an average latency of under 2 milliseconds, which represents a significant improvement and a good degree of scalability.

4.3 Android Performance

In order to complete the performance analysis of the Phoenix communication framework, we have conducted additional experiments using Android mobile devices. In particular, we have used a Google Nexus S smartphone as a reference, running Android 4.1.1 with a 1 GHz ARM CPU and 512 MBytes of RAM.

The first set of experiments was aimed at measuring the performance of the Phoenix communication framework when using wireless communication interfaces. Table 3 illustrates the average probe throughput and average probe latency that were measured using both WIFI and UMTS hardware interfaces. Note that probe messages were 280 bytes in size, throughput tests were repeated 25 times and the latency values represent the averages out of 100 measurements. Based on the results, we can assert that our Phoenix cluster is well capable of serving hundreds, if not thousands, of mobile devices running Phoenix clients.

The second set of experiments analysed the energy consumption derived from the use of the Phoenix in a mobile device. In particular, we conducted several experiments where the battery level of the smartphone was monitored during the execution of different communication routines. Table 4 shows the average duration of the battery under four different scenarios. In the first scenario, the device is in standby mode with the screen set to 5% brightness. In the second scenario, a publisher component generates 100 events/second while in the third scenario a subscriber receives 100 events/second. Finally, the fourth scenario combines the publication and the reception of 100 events/second. Based on the results, the impact of using Phoenix is moderate and its energy efficiency is fair.

5 Conclusion

In this paper, we have described the overall design and evaluation of a novel communication framework for ubiquitous systems. The framework is based on

the publish/subscribe communication model and has taken into account the requirements of ubiquitous systems. In particular, we have tried to design a communication framework that supports client mobility and provides a good degree of scalability. To that end, we have leveraged a routing algorithm [25] that provides optimal event routing and communication efficient client mobility.

The implementation of the Phoenix publish/subscribe framework has been motivated by the lack of suitable open source implementations that provide what we believe are key features of a communication infrastructure for ubiquitous systems, namely client mobility support and full integration with mobile devices. The reference implementation of Phoenix enables researchers and application developers to create ubiquitous systems and applications using wireless communication interfaces and nowadays common devices such as Android smartphones.

In addition, we have conducted both an empirical validation process and a thorough performance analysis and, based on the results, can assert that Phoenix is a valid and fairly scalable communication framework for ubiquitous systems.

Acknowledgements. The authors wish to note that this research has partially been supported by the Spanish Research Council, under grant TIN2010-17170, the Basque Government, under grants IT395-10 and S-PE11UN099, the Provincial Government of Gipuzkoa, under grant 2012-DTIC-000101-01, and the University of the Basque Country UPV/EHU, under grant UFI11/45.

References

1. Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R.E., Sturman, D.C.: An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In: ICDCS, pp. 262–272 (1999)
2. Burcea, I., Jacobsen, H.-A., de Lara, E., Muthusamy, V., Petrovic, M.: Disconnected Operation in Publish/Subscribe Middleware. In: Mobile Data Management, pp. 39–50. IEEE Computer Society (2004)
3. Caporuscio, M., Carzaniga, A., Wolf, A.L.: Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications. IEEE Transactions on Software Engineering 29(12), 1059–1071 (2003)
4. Carzaniga, A.: Architectures for an Event Notification Service Scalable to Wide-Area Networks. PhD thesis, Politecnico di Milano, Italy (1998)
5. Cugola, G., Jacobsen, H.-A.: Using Publish/Subscribe Middleware for Mobile Systems. Mobile Computing and Communications Review 6(4), 25–33 (2002)
6. da Costa, C.A., Yamin, A.C., Geyer, C.F.R.: Toward a General Software Infrastructure for Ubiquitous Computing. IEEE Pervasive Computing 7(1), 64–73 (2008)
7. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M.: The Many Faces of Publish/Subscribe. ACM Computing Surveys 35(2), 114–131 (2003)
8. Fiege, L., Gärtner, F.C., Kasten, O., Zeidler, A.: Supporting Mobility in Content-Based Publish/Subscribe Middleware. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 103–122. Springer, Heidelberg (2003)

9. Fiege, L., Zeidler, A., Gärtner, F.C., Handurukande, S.B.: Dealing with Uncertainty in Mobile Publish/Subscribe Middleware. In: Middleware Workshops, pp. 60–67. PUC-Rio (2003)
10. Huang, Y., Garcia-Molina, H.: Publish/Subscribe in a Mobile Enviroment. In: MobiDE, pp. 27–34. ACM (2001)
11. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M., Briggs, N., Braynard, R.: Networking Named Content. Communications of the ACM 55(1), 117–124 (2012)
12. Jayram, T.S., Khot, S., Kumar, R., Rabani, Y.: Cell-Probe Lower Bounds for the Partial Match Problem. Journal of Computer and System Sciences 69(3), 435–447 (2004)
13. Jerzak, Z.: XSiena: The Content-Based Publish/Subscribe System. PhD thesis, Technische Universität Dresden, Germany (2009)
14. Jerzak, Z., Fetzer, C.: Prefix Forwarding for Publish/Subscribe. In: Jacobsen, H.-A., Mühl, G., Jaeger, M.A. (eds.) DEBS. ACM International Conference Proceeding Series, vol. 233, pp. 238–249. ACM (2007)
15. Kale, S., Hazan, E., Cao, F., Singh, J.P.: Analysis and Algorithms for Content-Based Event Matching. In: ICDCS Workshops, pp. 363–369. IEEE Computer Society (2005)
16. Mühl, G., Fiege, L., Pietzuch, P.R.: Distributed Event-Based Systems. Springer (2006)
17. Mühl, G., Ulbrich, A., Herrmann, K., Weis, T.: Disseminating Information to Mobile Clients Using Publish-Subscribe. IEEE Internet Computing 8(3), 46–53 (2004)
18. Muthusamy, V., Jacobsen, H.-A.: Small Scale Peer-to-Peer Publish/Subscribe. In: Horrocks, I., Sattler, U., Wolter, F. (eds.) P2PKM. CEUR Workshop Proceedings, vol. 147. CEUR-WS.org (2005)
19. Muthusamy, V., Petrovic, M., Gao, D., Jacobsen, H.-A.: Publisher Mobility in Distributed Publish/Subscribe Systems. In: ICDCS Workshops, pp. 421–427. IEEE Computer Society (2005)
20. Muthusamy, V., Petrovic, M., Jacobsen, H.-A.: Effects of Routing Computations in Content-Based Routing Networks with Mobile Data Sources. In: Porta, T.F.L., Lindemann, C., Belding-Royer, E.M., Lu, S. (eds.) MOBICOM, pp. 103–116. ACM (2005)
21. Oki, B., Pflügl, M., Siegel, A., Skeen, D.: The Information Bus - An Architecture for Extensible Distributed Systems. In: SOSP, pp. 58–68 (1993)
22. Petrovic, M., Muthusamy, V., Jacobsen, H.-A.: Content-Based Routing in Mobile Ad Hoc Networks. In: MobiQuitous, pp. 45–55. IEEE Computer Society (2005)
23. Pietzuch, P.: Hermes: A Scalable Event-Based Middleware. PhD thesis, University of Cambridge, United Kingdom (2004)
24. Salvador, Z.: Client Mobility Support and Communication Efficiency in Distributed Publish/Subscribe. PhD thesis, University of the Basque Country, Spain (2012)
25. Salvador, Z., Larrea, M., Phoenix, A.L.: Phoenix: A Protocol for Seamless Client Mobility in Publish/Subscribe. In: NCA, pp. 111–120. IEEE Computer Society (2012)
26. Salvador, Z., Larrea, M., Phoenix, A.L.: Phoenix: A Protocol for Seamless Client Mobility in Publish/Subscribe. Technical Report EHU-KAT-IK-02-12, University of the Basque Country UPV/EHU (April 2012), `http://www.sc.ehu.es/acwlaalm/`

27. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications 8(4), 10–17 (2001)
28. Shen, Z., Tirthapura, S.: Faster Event Forwarding in a Content-Based Publish-Subscribe System through Lookup Reuse. In: NCA, pp. 77–84. IEEE Computer Society (2006)
29. Tarkoma, S.: Efficient Content-Based Routing, Mobility-Aware Topologies, and Temporal Subspace Matching. PhD thesis, University of Helsinki, Finland (2006)
30. Tarkoma, S., Kangasharju, J.: Handover Cost and Mobility-Safety of Content Streams. In: Boukerche, A., Leung, V.C.M., Chiasserini, C.-F., Srinivasan, V. (eds.) MSWiM, pp. 354–358. ACM (2005)
31. Tarkoma, S., Kangasharju, J.: Optimizing Content-Based Routers: Posets and Forests. Distributed Computing 19(1), 62–77 (2006)
32. Zeidler, A., Fiege, L.: Mobility Support with Rebeca. In: ICDCS Workshops, pp. 354–360. IEEE Computer Society (2003)