A Rule-Based Publish-Subscribe Message Routing System for Ubiquitous Computing

Yixin Jing[1], Dongwon Jeong[2], JinHyung Kim[1], and Doo-Kwon Baik[1]

[1] Department of Computer Science & Engineering, Korea University
Seoul, 136-701, Korea
{jing, koolmania}@software.korea.ac.kr, baikdk@korea.ac.kr
[2] Department of Informatics & Statistics, Kunsan National University
Gunsan, 573-701, Korea
djeong@kunsan.ac.kr

Abstract. The ubiquitous computing produces big volume of messages as the pervasive computability is deployed in large scale. As a middleware between the message producer and message consumer, message routing system enables backend systems to efficiently acquire the interested message. A widely adopted message routing mechanism is the content-based publish-subscribe framework. Based on this paradigm, we propose a rule-based system for supporting message routing in ubiquitous computing. The novel system features in the flexibility of the message computing, which is accomplished through a set of message operators. The message consumer could select the appropriate operator and specify the operating rule to get satisfying messages.

1 Introduction

Message routing system is a fundamental element in the ubiquitous computing architecture. The ubiquitous computing faces the challenges promoted by the explosion of the context information. RFID technology enables the product tracking at the instance-level rather than at the class-level [1]. Sensor network which is monitoring the environment constantly reports the current data and any changes. Countless mobile devices generate huge numbers of unanticipated events. All of this information, including event and data, gathered from the physical world is defined as messages. The hardware or software which generates the message is so called the message producer. The message generated from the producer is called raw message, which flows from the message producer to the backend systems or applications to get treatment. Those systems or applications where the message sinks are called as message consumer.

However, the message generated from the message producer is usually of interest not only to a single consumer, but to a diverse set of consumers across an organization and its business partners. The message data must thus be broadcasted to the entities that indicated an interest in the data. On the other hand, common to all consumers that make use of the message is the desire to receive filtered and aggregated message rather than raw streams of message. Different consumers are

H.Y. Youn, M. Kim, and H. Morikawa (Eds.): UCS 2006, LNCS 4239, pp. 1 – 12, 2006.

however interested in a different subset of the total message captured based on the message producer. The entity which deals with the message disseminating, filtering and aggregating is a kind of middleware decoupling the message producer and consumer. The middleware like that is named as message broker. Message producer, broker and consumer form the basic elements of the message routing system, see Fig. 1.

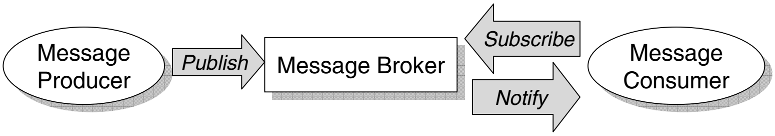


Fig. 1. Message routing system

Publish-subscribe (pub-sub for short) is an important paradigm for asynchronous communication between the message producers and message consumers. In the pub-sub paradigm, subscribers (message consumer) specify their interests in certain message conditions, and will be notified afterwards of any message fired by a publisher (message producer) that matches their registered interests. Pub-sub systems can be characterized into several broad types based on the expressiveness of the subscriptions they support. Content-based pub-sub is a general and powerful paradigm, in which subscribers have the added flexibility of choosing filtering criteria along multiple dimensions, using thresholds and conditions on the contents of the message [2].

In the ubiquitous computing, pub-sub should be an easy process which enables un-expert subscriber to progress smoothly. However the prevailed content-based pub-sub is designed for distributed network which result in a complicated message pub-sub process. To overcome this weakness, we propose a novel content-based message routing system. The system features in a rule-based approach to cope with the message routing. Compared with other approaches, the proposed system has a clear guideline to ease the subscription. The paper is organized as the follows. Section 2 introduces the related work. Section 3 explains a XML based message expression. The rule-based routing method is present in the Section 4. Section 5 proposes the architecture of the message routing system and relative solutions. We conclude the paper in the Section 6.

2 Related Work

Publication-subscription message routing mechanism has been evolving for years [3]. Channels [4, 5] are the first generation of publish/subscribe systems. A notification is published to a specific channel specified by the publisher and delivered to all consumers that have subscribed to this channel. However, applications consuming context information do not need to maintain communication channels with individual message producer, but can simply specify which message they are interested in by submitting subscriptions to the message broker.

Subject-based publish/subscribe systems [6-8] associate with each notification a single subject that is usually specified as a string. Subjects can be arranged in a tree by using a dot notation. The drawback of this approach in ubiquitous computing is that the any change of the subject will cause the rearrange of the subject tree. In ubiquitous computing, the potential message consumers, namely subjects, are hard to anticipate, which make the messaging servers unable to afford the high cost brought by frequent change to the subject tree.

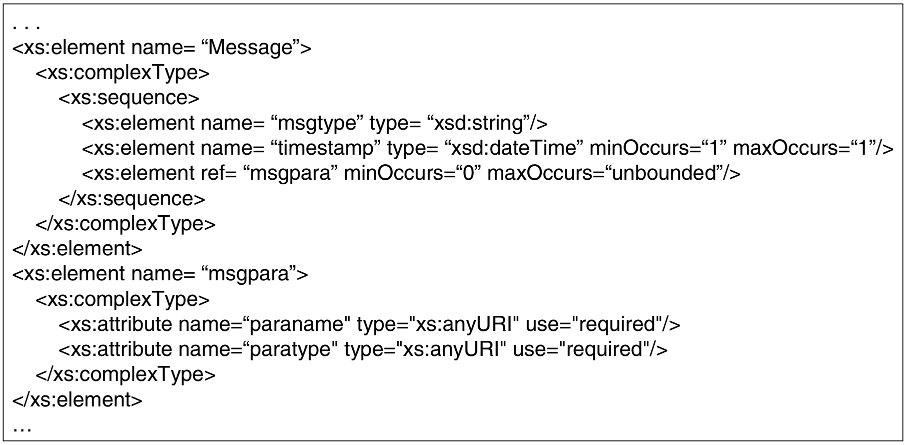
Content-based pub-sub is a more general and powerful paradigm, in which subscribers have the added flexibility of choosing filtering criteria along multiple dimensions, using thresholds and conditions on the contents of the message, rather than being restricted to (or even requiring) pre-defined subject fields. Clearly, it is the most flexible mechanism adaptive to the dynamic characters of ubiquitous computing. The research on this area produce considerable outputs including Elvin [9], Gryphon [10-12], Keryx [13], Siena [14, 15], Le Subscribe [16, 17], Jedi [18], the CORBA Notification Service [19], and Rebeca [20]. These routing systems are designed for the environments which have high demand for treating complicated message types, such as XML document. Considering the comparative simple message, an easy-to-handle content-based message routing is more suitable for ubiquitous computing. Motivated by this goal, we propose a novel rule-based message routing for efficiently delivering message to interested destinations.

3 Message Expression

We use a XML based expression to represent the message, including the input and output of the message broker. Although XML encoding adds substantial overhead, XML provides more opportunities for structure. Parsing every incoming message, constructing every outgoing message, and transmitting information in the verbose XML format, may reduce message throughput and adds latency along the event flow. However, compressing the XML reduces bandwidth consumption [21], and enhances interoperability and extensibility. In addition, most messages in ubiquitous computing have a simple structure, we exploits an XML based message representation in our system.

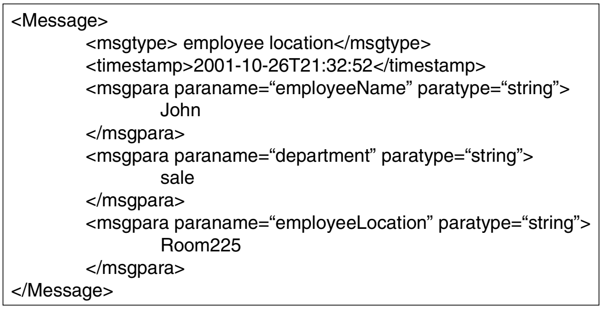
The message could be an event, context information from sensors, or a tag data read by RFID readers. We assume that before the message arrive the message broker, it could get formatted by message producers. Or, after the source message arrival, it could be formatted within the message broker. The issue of how to transform the heterogeneous message to the expression defined in this paper is out of scope of this paper. Part of the XML schema of the message is shown as the Fig. 2.

The message has a root element <Message>. The message information is classified into two components, viz. the basic information (such as message type and timestamp) and the message detail. The message detail is contained in <msgpara> element, which attributes consist of parameter name and data type. Fig. 3 illustrates a message example. The example is generated by sensors detecting the employee ID cards, which tells his/her location in the company.



```
. . .
<xs:element name= “Message”>
   <xs:complexType>
      <xs:sequence>
         <xs:element name= “msgtype” type= “xsd:string”/>
         <xs:element name= “timestamp” type= “xsd:dateTime” minOccurs=“1” maxOccurs=“1”/>
         <xs:element ref= “msgpara” minOccurs=“0” maxOccurs=“unbounded”/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name= “msgpara”>
   <xs:complexType>
      <xs:attribute name=“paraname" type="xs:anyURI" use="required"/>
      <xs:attribute name=“paratype" type="xs:anyURI" use="required"/>
   </xs:complexType>
</xs:element>
…
```

Fig. 2. Message schema based on XML



```
<Message>
            <msgtype> employee location</msgtype>
            <timestamp>2001-10-26T21:32:52</timestamp>
            <msgpara paraname=“employeeName” paratype=“string”>
                        John
            </msgpara>
            <msgpara paraname=“department” paratype=“string”>
                        sale
            </msgpara>
            <msgpara paraname=“employeeLocation” paratype=“string”>
                        Room225
            </msgpara>
</Message>
```

Fig. 3. The message example. John, who belongs to the sale department, is in the room 225 now.

4 The Rule-Based Message Routing

In this section, we introduce the rule-based message routing. In the proposed routing approach, we adopt message operator to route the message flow according to the subscription. We first define the concept of message operator, then, explain how to define a rule based on the message operator.

4.1 Message Operators

To provide a message broker which is capable of meeting the application requirement, we define a set of message operator to complete the primitive message treatment. Three kinds of message operator are defined (see Fig. 4).

The message operator enables the message broker to deal with flexible message routing. The raw message is not generated for the particular applications (message consumer). The context information included in the message from the message

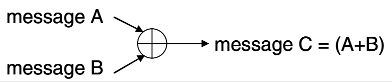
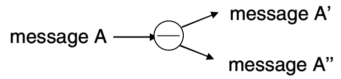
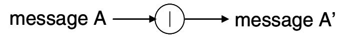
Operator	Explanation	Example
Merge	Merge several messages into one	
Split	Split one message into several smaller piece	
Transform	Transform the message information, such as the measurement	

Fig. 4. Message operators

producer is probably not sufficient for the application to execute given process. To solve this problem, the application has to gather all necessary messages itself. However, this expectation is not applicable in ubiquitous computing. Since in most cases, applications are designed and developed independently from the sensor level. If the application is tightly coupled with the sensor level, it looses flexibility when used in different scenarios. In this case, the message operator in a message broker is required to enrich the raw message and possibly to inquire about more information from several context sensors. Consider the scenario in the Fig. 5 where two monitor systems are using sensors to detect the location of the employee and car in the parking lots. The location information of each employee and each car is constantly sent out through message flow. At the backend, an alert application is running by analyzing the distance between the car and its owner. If the distance exceeds a given maximum number, a car-stealing alert would be sent to the owner. To support the application, the location of the employee as well as that of the car should be merged into one message and delivered to the alert application.

On the contrast, raw messages may contain a lot of information that is not pertinent to the application. When these potentially irrelevant messages are delivered to the destination application without being tailored, the redundant data increase the network overload. In addition, when this meaningless data arrives the application, extra computing resources are consumed to digest the volume of data. In Fig. 5, two working management systems in different departments are interested in the different employees. The split operator can filter the messages, and pass the messages to the different department systems.

In some other cases, the encoding of the message information does not match the application requirement. For example, in Fig. 5, the RFID reader of the car monitor system can receive 64bit EPC of a car while the backend parking management system can only process 96bit EPC. To overcome the code set gap, a message transform operator could help to change the encoding.

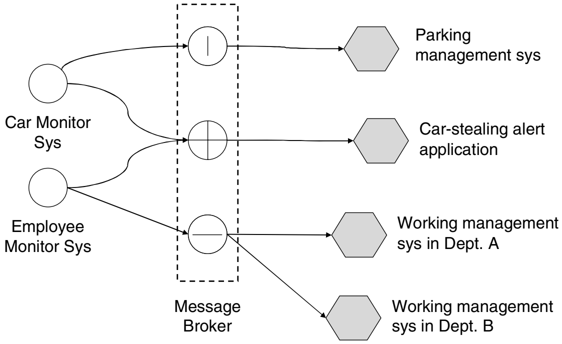

Parking management sys
Car-stealing alert application
Car Monitor Sys
Employee Monitor Sys
Working management sys in Dept. A
Message Broker
Working management sys in Dept. B

Fig. 5. Scenario of the message operator utilization

4.2 Message Operator Definition

Each kind of message operator has the input and out put message. In the operator, some rules have to be pre-defined. When the incoming messages meet the condition of the rules, the messages are merged, split, or transformed following the rules. The definition of operator is actually a XML file which restricts the input message, output message and the corresponding rules. The general structure of the operator is shown in Fig. 6. Operator information includes basic description such as creation date and subscriber. Input message depicts the message types which are going to be operated. The output message is defined for producing the message in a given format to satisfy the message consumer. In addition, the output message component contains the information of definition where the generated message be delivered. One output message can be sent to multiple consumers. Rule is a key component where the operator processor can obtain necessary conditions to check whether the input messages meet the operator requirement or not. Within the rule, a set of functions could be executed and finally return a Boolean value.

We take the merge operator for car-stealing alert application as an example. The employee location message was illustrated in the Fig. 3. Likewise, the message of the car location is defined in the similar way, which includes the type information of the car and the name of the car's owner. Fig. 7 shows the definition of the merge operator.

The merge operator defines the format type of output message as well as the rule. The operator is specified by the message subscriber, in this example, namely the alert application. During the subscription (discussed later), the alert application has to define a target message format, mycar, and specify the rule. Under the element <rule>, different conditions could be defined. Condition <equal> restricts that the value of the two elements should be the same. Only when the employee name is the same as the owner of the car, the input messages could be merged into one according to the format defined under <msgformat>. XPath [22] expressions are adopted to locate the target elements for comparison, and to fill the merged message with the value from input messages. Finally the merged message is sent to the target address depicted in the <destination> element.

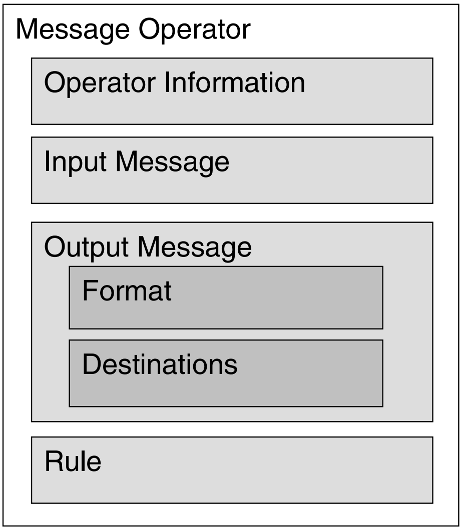


Fig. 6. The construction of operator

```
<Merge id=“merge001”>
        <input>
                <msgtype>employee location</msgtype>
                <msgtype>car location</msgtype>
        </input>
        <output>
                <msgformat>
                        <msgtype>mycar</msgtype>
                        <timestamp/>
                        <msgpara paraname=“employeeName” paratype=“string”
                                valuefrom=“employee location ://msgpara[@paraname=‘employeeName’]”/>
                        <msgpara paraname=“employeeLocation” paratype=“string”
                                valuefrom=“employee location ://msgpara[@paraname=‘employeeLocation’]”/>
                        <msgpara paraname=“carLocation” paratype=“string”
                                valuefrom=“car location ://msgpara[@paraname=‘carLocation’]”/>
                </msgformat>
                <destinations>
                        <destination>192.168.0.15:80</destination>
                </destinations>
        </output>
        <rule>
                <equal>
                        <element>employee location://msgpara[@paraname=‘employeeName’]</element>
                        <element>car location://msgpara[@paraname=‘carOwner’] </element>
                </equal>
        </rule>

</Merge>
```

Fig. 7. Operator for merging context message of the employee and car

4.3 Message Operator Customization

The message operators provide a very flexible approach for subscriber to customize the subscription. This flexibility is carried out by two perspectives.

First, the structure of the message operator is easily to extend. XML-based message operator is easy to understand for non-expert users. Additionally, the message broker could provide plenty choices of functions for message subscribers to customize the operator. Within the <rule> element of operator, more predefined functions could be provided. For example, the subscriber can make the judgment on the value with <isString>, or can compare the values carried by the input messages with <greaterThan>. Whatever functions are adopted, the returned result should be a boolean value, i.e. true or false.

Second, the combination of the message operators significantly magnifies the subscriber capability of selecting interesting messages. The message operator does not must to be utilized stand alone. The message subscriber could combine the different operator types to get a more precise output . For example, a merged message can be split again depending on different requirements, or can be transformed to another encode to feed special subscriber. In this case, the message routing system needs to manage the operators in one subscription to ensure they can cooperate correctly. This process in dealt with during message subscription. When the message consumers attempt to use multiple operators, the only thing they need to do is to alter the destination denoted in the operator to other operators' ID instead of IP address.

5 Message Routing System

The section describes the implementation of the message routing system. First we will introduce the architecture of the system. Second, we present the algorithm of the message merge operator.

5.1 System Architecture

In this section we present the primitive message routing system and explain how to implement the capability enumerated in the previous section. Fig. 8 show the architecture of the system.

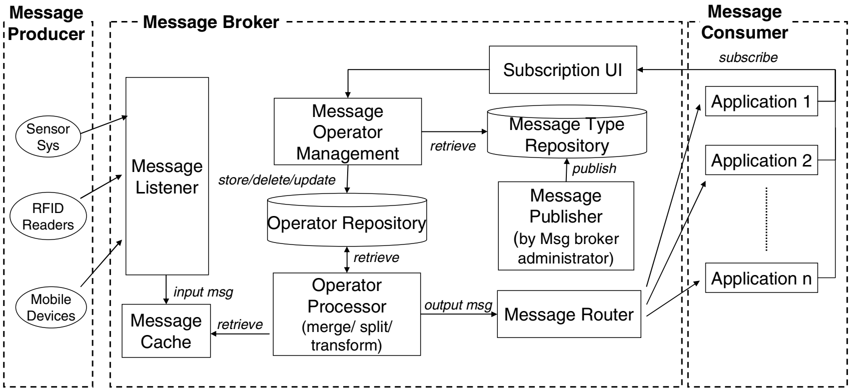


Fig. 8. The architecture of the message routing system

The entire system can be partitioned into three functions, message publish, subscribe and route. Message publisher is responsible for publishing the different types of incoming messages to a repository. The Subscription UI is an interface through which the message consumers can create, delete or update their subscription. This process is accomplished through manipulating the operator XML files. First, the message consumer has to retrieve the current message type published by the message broker. After selecting the interested message type, the message consumer has to decide the operator type. As the operator type varies, the Message Operator Manager provides different UI styles for the message consumer to complete the subscription. In the Subscription UI, the message consumer defines the message format for the output message as well as the rules. Based on the interested message content, the Message Operator Manager can formulate the subscription into a XML file and store the file in the Operator Repository. The operator file stored in the Operator Repository is referenced by the Operator Processor. The incoming messages are stored in the Message Cache first. Responding to the input message, the Operator Processor reads the predefined operator in the repository, then merges, splits or transforms the message under the guideline specified in the operator. Finally, the output message is delivered to the corresponding message consumer through a Message Router.

5.2 Algorithm for Merge Operator

Although all of three message operators have distinguished algorithms, message operator is the most complicated one due to its multiple incoming. This subsection takes the merge operator to investigate how system deals with the subscription which requires a message merging.

Two major factors influence the performance of merge operator. One is the interval time t_o at which the message subscriber wants the output message to be delivered out. The other one is the interval time t_i that the input messages arrives the message cache. Obviously when $t_o < t_i$, the message subscriber could not be sufficiently feed. Therefore, when the message broker receives the subscription, the broker has to acquire t_o from the subscriber to assure that the desired output message could be delivered in time. According to the cache size and operator processor's performance ability, an appropriate memory space is allocated to cache the constantly input messages. In our system prototype, we adopt a matrix to cache the input messages. The row number of matrix, r, equals the number of the input message going to be merged. The column number is decided by the following function.

$$c = \frac{T_h}{t_o} \tag{1}$$

Where T_h is the period time, in which the incoming messages will be stored, otherwise will be considered be expired. For example, if the system decides to save the messages received in recent 300 seconds and the message consumer wants to get merged message every 60 seconds, the c = 300/60 = 5. If two kinds of input messages are candidate to be merged, the matrix looks like that shown in Fig. 9.

$$M = \begin{array}{cccccc} & t_1 & t_2 & t_3 & t_4 & t_5 \\ m_1 & \circ & \circ & \circ & \circ & \circ \\ m_2 & \circ & \circ & \circ & \circ & \circ \end{array}$$

Fig. 9. Storage matrix for incoming message

The object saved in each element of the matrix is the message content for sake of the small message size. During each 5 seconds, the cache saves the newest input message in one matrix column according to the message type. In the next 5 seconds, the cache will save the input messages to the next matrix column. When the process for saving messages meets the fifth column, it will re-begin from the first column. The old messages in the first column are replaced, i.e. treated as expired.

The processing for reading the messages from the matrix is synchronous with the process for saving. Thus, the two processes will not conflict as long as they don't access to the same column at the start time. The algorithms of the saving and reading are present in Fig. 10 a) and b) respectively.

```
Read $t_0$ from the subscription;
Create the message matrix;
int $i$ = 0;
Create a timer, within which
  $i=(i+1) mod c$ when every $t_0$ pass by;
Start the timer;
While (true) do {
    get input message;
    get message type $j$;
    save message to M[j][i];
}
```

a) pseudo code for saving

```
int $x$ = 0;
Create a timer, when every $t_0$ pass
  by, do{
    x = (x=1) mod c;
    get input messages in column $x$;
    merge the messages;
    send out the merged message;
}
```

b) pseudo code for merging

Fig. 10. Algorithms of the saving and retrieve messages

6 Conclusion

In this paper we contributed a rule-based message routing system for ubiquitous computing. The mechanism is based on the pub-sub framework. The rule for routing the message is defined by subscribers during the subscription. Under the guideline of the rule, the incoming message could be computed by utilizing the merge operator, split operator or transform operator, to meet the message consumer interests. The system benefits from the flexibility of the message operator. In essence, message operator is an atomic message computing unit. More sophisticated message computing could be achieved by combining the different operators. The prototype implementation and experiment would be covered in our future work.

References

1. Floerkemeier, C., and Lampe, M.: RFID Middleware Design: Addressing Application Requirements and RFID Constraints. In: The 2005 Joint Conference on Smart Objects and Ambient Intelligence. ACM International Conference Proceeding Series, Vol. 121. (2005) 219–224
2. Cao, F., and Singh, J.P.: Efficient Event Routing in Content-Based Publish-Subscribe Service Networks. In: The INFOCOM 23rd Annual Joint Conference of the IEEE Computer and Communications Societies. Vol. 2. (2004) 929-940
3. Mühl, G.: Generic Constraints for Content-Based Publish/Subscribe. In: The 9th International Conference on Cooperative Information Systems (CoopIS). Lecture Notes in Computer Science, Vol. 2172. (2001)211-225
4. OMG: CORBA event service specification. OMG Document formal/94-01-01(1994)
5. Sun: Distributed event specification (1998)
6. Oki, B., Pfluegl, M., Siegel, A., and Skeen, D.: The Information Bus-An Architecture for Extensible Distributed Systems. In: ACM Symposium on Operating Systems Principles. (1993) 58–68
7. Sun: Java message service specification 1.0.2 (1999)
8. TIBCO Inc.: TIB/Rendezvous. http://www.tibco.com/. (1996)
9. Segall, B., Arnold, D., Boot, J., Henderson, M., and Phelps, T.: Content Based Routing with Elvin4. In: AUUG2K (2000)
10. Aguilera, M., Strom, R., Sturman, D., Astley, M., and Chandra, T.: Matching Events in a Content-Based Subscription System. In: The 18th ACM Symposium on Principles of Distributed Computing (PODC) (1999)53-61
11. Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. E., and Sturman, D. C.: An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In: The 19th IEEE International Conference on Distributed Computing Systems (1999) 262
12. Opyrchal, L., Astley, M., Auerbach, J., Banavar, G., Strom, R., and Sturman, D.: Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In: Sventek, J., Coulson, G. (eds.): Middleware. Lecture Notes in Computer Science, vol. 1795 (2000)185–207
13. Wray, M., Hawkes, R.: Distributed Virtual Environments and VRML: An Event-Based Architecture. In: The 7th International WWW Conference (1998)43-51
14. Carzaniga, A., Rosenblum, D., and Wolf, A.: Content-Based Addressing and Routing: A General Model and its Application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, USA (2000)
15. Carzaniga, A.: Architectures for an Event Notification Service Scalable to Wide-area Networks. PhD thesis, Politecnico di Milano, Milano, Italy (1998)
16. Fabret, F., Llirbat, F., Pereira, J., and Shasha, D.: Efficient Matching for Content-Based Publish/Subscribe Systems. Technical report, INRIA (2000)
17. Pereira, J., Fabret, F., Llirbat, F., and Shasha, D.: Efficient Matching for Web-Based Publish/Subscribe Systems. In: Etzion, O., Scheuermann, P. (eds.): The International. Conference on Cooperative Information Systems (CoopIS). Lecture Notes in Computer Science, Vol. 1901 (2000) 162-173

18. Cugola, G., Di Nitto, E., and Fuggetta, A.: Exploiting an Event-Based Infrastructure to Develop Complex Distributed Systems. In: The 1998 International Conference on Software Engineering. IEEE Computer Society Press / ACM Press (1998) 261–270
19. OMG: Corba Notification Service. OMG Document telecom/99-07-01 (1999)
20. Fiege, L., and Mühl, G.: Rebeca Event-Based Electronic Commerce Architecture. http:// www.gkec.informatik.tu-darmstadt.de/rebeca (2000)
21. Snoeren, A. C., Conley, K., and Gifford, D. K.: Mesh Based Content Routing using XML. In: ACM SIGOPS Operating Systems Review. ACM Press, Vol. 35 (2001)160-173
22. W3C: XML Path Language (XPath) 2.0, W3C Candidate Recommendation. http:// www. w3.org/TR/2005/CR-xpath20-20051103/ (2005)