AGVENT: AGENT EVENTS

Ozgur Koray SAHINGOZ
Air Force Academy
Computer Engineering Department
Yesilyurt, Istanbul, TURKEY
sahingoz@hho.edu.tr

Nadia ERDOGAN
Istanbul Technical University
Electrical-Electronics Faculty
Computer Engineering Department, Ayazaga
80626, Istanbul, TURKEY
erdogan@cs.itu.edu.tr

ABSTRACT
A publish/subscribe system is a middleware communication service that delivers messages from a sender to one or more receivers using the preferences expressed by those receivers, rather than relying on an explicit destination address set by the sender. The historical development of publish/subscribe systems has followed a line which has evolved from channel-based systems, to subject-based systems, next content-based systems and finally object-based systems. In this paper, we propose a new model for agent based distributed events systems, the Agvent system, which combines the advantages of publish/subscribe communication and mobile agents into a flexible and extensible distributed execution environment. The Agvent system exploits a two-leveled mobile agent as mediators between participants of an event based distributed system.

1. INTRODUCTION
In client/server systems two roles exist: A component acts as a client if it requests data or functionality from another component; it acts as a server if it responds to a client's request. Moreover, a client is blocked after it has issued a request, until the corresponding reply arrives. RPC (remote procedure call) [1] and more recently object-oriented successors like RMI (remote method invocation) [2] automated request/reply-based inter-action in such a way that calling a remote procedure/method is almost identical to the local case. For example, arguments and return values are automatically marshalled and unmarshalled.

With the use of mobile and/or large-scale systems, the need for asynchronous, loosely coupled and point to multipoint communication pattern arises. The *publish/subscribe* paradigm serves these needs. Event models are application independent infrastructures that satisfy communication requirements of such systems. Event-based communication generally implements the publish/subscribe model, as shown in Figure 1. A publish/ subscribe system consists of a set of clients that asynchronously exchange notifications, decoupled by a dispatch service, which is interposed between them. Clients can be characterized as producers or consumers. *Producers (publisher)* publish notifications, such as current stock quotes, and *consumers (subscribers)* subscribe to notifications by issuing subscriptions, which are essentially stateless message filters.

To receive event data, subscribers register to an event service with the definitions of the events they are particularly interested in. A definition can include a simple subscription message, a subscription message with filtering on events or a subscription message for composite events [3].

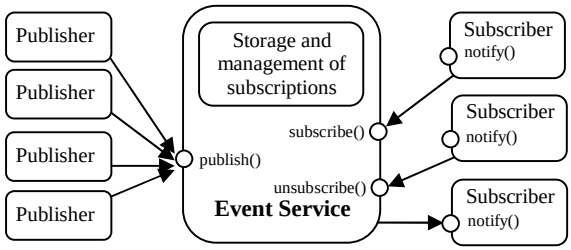

Figure 1. Publish/Subscribe model

The benefits of publish/subscribe protocol make it first choice for implementing information-driven applications. For example, publish/subscribe is well suited for information dissemination applications like news delivery, stock quoting, electronic commerce[4] air traffic control [5], and dissemination of auction bids [6]. The use of publish/subscribe techniques has also been described in the areas of mobile agents [7], workflow systems [8], and process control systems [9].

In this paper, we present an agent based distributed event system, the Agvent system, which exploits mobile agents as mediators between publishers and subscribers of events. The Agvent system implements the publish/subscribe protocol, thus enabling many-to-many interaction of loosely coupled entities. It also allows

publishers and subscribers to dynamically connect and disconnect from the system, a capability that extends the flexibility of the working environment.

The rest of this paper is organized as follows. Firstly, we present a classification of publish/subscribe systems with references to related work. After that, Section 3 introduces the design decisions and framework of Agvent system. Finally, our conclusions are presented in Section 4.

2. PUBLISH/SUBSCRIBE SYSTEMS

Publish/subscribe communication systems are characterized by the complete decoupling of producers (publishers) and consumers (subscribers) of data. In this model, receivers of messages express their interest by subscribing to a class of events and they are asynchronously notified if a sender publishes an event which matches their subscription. In this way, the model allows a flexible n-to-m communication among communicating parties. Publish/subscribe systems can be present in the application domain of software systems, networked interactive games, news, internet-based trading, etc. They can be classified into four groups according to their subscription mechanism. Each one is discussed in detail below.

2.1. Channel-based systems

The simplest subscription mechanism is what is commonly referred to as a channel. Subscribers subscribe or listen to a channel. Applications explicitly notify the occurrence of events by posting notification to one or more channels. The part of an event that is visible to the event service is the identifier of the channel to which the event has been sent. Every notification posted to a channel is delivered by the event service to all the subscribers that are listening to that channel. Channels can be implemented efficiently because they can easily be mapped to multicast groups, but they have some inherent disadvantages. Firstly, the expressiveness, i.e., the filtering capability, of channels is rather limited because notifications can only be classified with respect to a number of channels. Secondly, channels are inflexible and inhibit changes. If the assignment of notifications to channels changes, both producer and consumers may have to be changed. Finally, producers and consumers are not fully decoupled because the producer decides into which channel(s) a notification is to be published.

The abstraction of the channel is equivalent to the one given by a mailing list. A user sends an e-mail to an address, and message is forwarded to those who have registered to that mailing list. CORBA Event Service [10] adopts a channel-based architecture. Another widely used channel based model is the Java Delegation Event Model [11], which encapsulates events from the platform's Graphical User Interface

2.2. Subject-based systems

Some systems extend the concept of a channel with a more flexible addressing mechanism that is often referred to as subject-based[1] addressing. In this case, an event notification consists of two different parts: a well-known attribute, the subject, which determines the address, which is followed by the remaining information of the event data. The main difference with respect to a channel is that subscriptions can express interest in many subjects/ channels by specifying some form of expression to be evaluated against the subject of a notification. This implies that a subscription may define a set of event notifications, and two subscriptions may specify two overlapping sets of notifications. This, in turn, implies that one event may match any number of subscriptions.

JEDI [8] adopts the subject-based subscription mechanism. In JEDI, an event is given in the form of a function call; where the first string is the function/event name followed by parameters, e.g., “print (tez.doc, myprinter)”. Each event is labeled with a subject. Subscriptions are specified with an indication of the subject of interest. Notice that the subject-based approach is a variation of the channel based concept, as the rest of the event data except for the subject is content-free. The subject can be a list of strings in a hierarchical form, over which it is possible to specify filters based on a limited form of regular expressions. For example, the filter “economy.exchange.*.*HOL” (as a subtree structure) will select all the notifications whose subject contains economy in first position followed by exchange in second position, any string in third position, and a fourth string that ends with the string “HOL”.

The above examples show that subjects provide more powerful notification selection than channels. Nevertheless, subjects have a number of drawbacks. Firstly, they still have a limited expressiveness. With subjects it is possible to have a subject for each single stock, but what if the user is interested in the stock price only if it rises above a certain limit? Secondly, subjects are only suitable to divide the notification space with respect to one dimension. Finally, changes to the subject tree can require major application fixes.

2.3. Content-based systems

By extending the domain of filters to the whole content of notifications, some researchers obtain another class of subscriptions called content-based [12]. Content-based subscriptions are conceptually very similar to subject-based ones. However, since they can access the whole structured content of notifications, an event server gives more freedom in encoding the data upon which filters can be applied and that the event service can use for setting up routing information. The increase in

[1] Some authors use “topic-based subscription” instead of “subject-based subscription”

expressiveness allows the delivery of uninteresting notifications to be reduced or even to be avoided. In particular, this is important for applications that run on mobile devices having limited processing power and network bandwidth.

Examples of event systems that provide this kind of subscription are Yeast [12] (uses a centralized structure) and SIENA[13]. In SIENA, an event notification is a set of attributes in which each attribute is a triple, as in "attribute = (name; type; value)". Attributes are uniquely identified by their name. An event filter defines a class of event notifications by specifying a set of attribute names and types and some constraints on their values, e.g., "attr filter = (name; type; operator; value)".

2.4. Type (Object) -based systems

Type based publish/subscribe model [14], proposed by Eugster, is a new model of subscription that has been developed to access event data in a more structured manner. Events are often viewed as low-level messages and a predefined set of such message types are offered by most systems, providing very little flexibility. To overcome this deficiency, type-based publish/subscribe mechanism manipulates events as objects, called *obvents*. The core idea underlying this integration consists in viewing events as first class citizens, and subscribing to these events by explicitly specifying their type. So an application-defined event data can be used in the event system.

Type-based publish/subscribe has several advantages over other publish/subscribe variants. By reusing the type scheme of the language to classify message objects, type-based publish/subscribe avoids any unnatural subscription scheme and provides for a seamless integration of a publish/subscribe middleware with the programming language. The knowledge of the type of message objects also enables the generation of static filters for content-based publish/ subscribe from dynamically defined requirements.

While surveying the progression of the publish/subscribe systems, we have concluded that the next step of the evolution should be based on agents. Therefore, in this paper, we propose a new model for a distributed event system , the Agvent System, which uses mobile agents (*agvents* - *ag*ent e*vents*) as event data in publish/subscribe protocol. In the next section, we explain the advantages of the proposed system and describe the major design decisions along with its framework.

3. AGVENT SYSTEM

The Agvent System is an agent based distributed event system whose framework is shown in Figure 2. The system consists of three main components: `publishers` that submit information to the system, `subscribers` that express their interest in specific types of information and a `dispatch service`, which is responsible for dispatching the incoming agvents. Our goal is to combine two developing technologies, mobile agents and publish/subscribe system, in order to benefit the advantages of both. The advantages derived from the general characteristics of the publish/subscribe protocol are the following [15].

- **Space Decoupling:** producers do not need to address consumers and vice versa. Instead, consumers simply specify the notifications they are interested in. This loosely coupled approach facilitates flexibility and extensibility because new consumers and producers can be added, moved, or removed easily.
- **Flow Decoupling:** communication is asynchronous, thereby removing the disadvantages and in flexibility of synchronous communication described above.
- **Time Decoupling:** producers and consumers do not need to be available at the same time. This means that a subscription causes notifications to be delivered even if producers join after the subscription was issued.

We want to combine the general properties of software agents to this system [16].

- **Autonomy:** Agents should be able to perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and they should have a degree of control over their own actions and their own internal state
- **Social Ability:** Agents should be able to interact, when they deem appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities. This requires that agents have, as a minimum, a means by which they can communicate their requirements to others and an internal mechanism for deciding when social interactions are appropriate (both in terms of generating appropriate requests and judging incoming requests).
- **Reactivity:** Agents should perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined) and respond in a timely fashion to changes, which occur in it.
- **Proactiveness:** Agents should not simply act in response to their environment; they should be able to exhibit opportunistic, goal-directed behavior and take the initiative where it is appropriate.

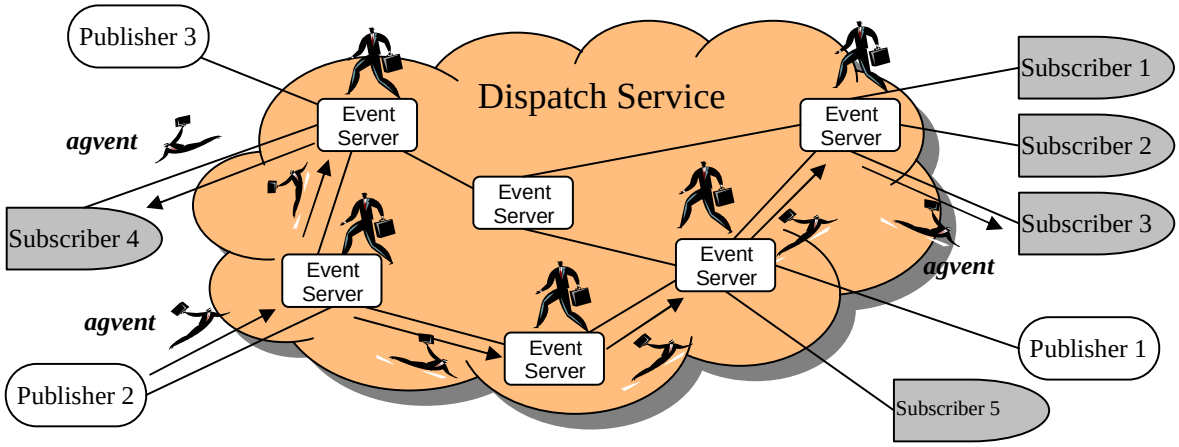

Figure 2. Framework of the Agvent System

The system applies the capabilities of agents stated above to real-world "entities" on whose behalf they operate, whether that entity be a person, a place, or even a less concrete notion like an organizational group.

The Agvent system integrates mobile agent technology with publish/subscribe communication to reach a new model for agent based distributed system. The Agvent System differs from other distributed event systems with its distinct characteristics that are described below.

a) *Autonomous Events.* Events are not viewed as simple messages. In most event systems, events are defined as low-level messages, which consist of record-like structures, list of strings, tuple-based structures, or etc. In type-based systems, events are defined as objects and viewed as first class citizens. Nevertheless, they are not autonomous.

   In the Agvent System, events are defined as mobile agents that have their own goals, beliefs and behaviors, which are loaded to the agvent when they are created.

b) *Agvent Based Subscription:* In most distributed event systems, Subscribers register on a channel or on a specific topic.

   In the Agvent System, subscribers register on agvent types. For example, a subscriber can register on an agvent, which is an instance of "Agv_type1" class.

c) *Information Hiding.* In previously developed event systems, event servers can access (has to) the content of the published event data before it can dispatch these data to the registered targets.

   In the Agvent System, the published agvent itself searches the knowledge base of the event server, selects the registered subscribers, clones itself and sends each agent clone to a subscriber on the list. Therefore, the role of the event server is reduced and an event server can be developed easily.

d) *User/Application defined event (agvent) types.* Distributed event systems generally use predefined event types. Therefore, to add a new event type you have to make programmatic changes in dispatch service, publisher, and subscriber sites.

   In the Agvent System, a publisher creates its own agvent type and sends it to the dispatch service. Once an agvent type is defined, subscribers can subscribe on agvents of that type.

Participants of the Agvent System follow two different models: the publication model and the subscription model.

3.1. Publication model:

The publication model defines data model for publishable event data. In most distributed event systems, this model should classify services according to the following parameters:

- *structure:* characterizes the structure of notifications. Typical publications can be classified as unstructured, lists of strings, record-like structures with positional or name-based identification of attributes, recursive structures, such as LISP expressions or XML documents, and composite publications, made of digests of other publications
- *types:* predefined domains of values. Typical type classifications would be binary or string, simple atomic types (such as integers, dates, booleans), and typed structures, that is, structures whose combination of fields constitute a type in itself.
- *limits:* total byte size, number of attributes, limits for types (string length, integer sizes or ranges of values), and number and depth of sub-structures.

The Agvent System uses mobile agents [17, 18] for searching, retrieving and dispatching event data. There are at least seven main benefits of using mobile agents.

a) *They reduce the network load.* Mobile agents allow users to package a conversation and dispatch it to a destination host where interactions take place locally.

b) *They overcome network latency.* Mobile agents offer a solution, because they can be dispatched from a central controller to act locally and execute the controller's directions directly.
c) *They encapsulate protocols.* Mobile agents, on the other hand, can move to remote hosts to establish channels based on proprietary protocols.
d) *They execute asynchronously and autonomously*. Mobile devices often rely on expensive or fragile network connections. Tasks requiring a continuously open connection between a mobile device and a fixed network are probably not economically or technically feasible. To solve this problem, tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the agents become independent of the process that created them and can operate asynchronously and autonomously. The mobile device can reconnect at a later time to collect the agent.
e) *They adapt dynamically.* Mobile agents can sense their execution environment and react autonomously to changes.
f) *They are naturally heterogeneous.* Mobile agents are generally computer- and transport- layer-independent (dependent on only their execution environments), they provide optimal conditions for seamless system integration.
g) *They are robust and fault-tolerant*. Mobile agents' ability to react dynamically to unfavorable situations and events makes it easier to build robust and fault-tolerant distributed systems.

Publishers are responsible of creating agvents. When a publisher decides to create a new agvent, it defines the agvent's goals, beliefs and behaviors. After that, it sends this agvent to the Dispatch Service over an Event Server to which it is connected (as depicted in Figure 1). Event Servers and publishers provide a platform for incoming agvents to run autonomously.

After an agvent is activated on the Event Server, it

- checks the knowledge base of the event server,
- selects the targets (event servers or subscribers)
- creates its clones for each target
- send these clones to the targets

When an agvent reaches a subscriber site, it

- checks the knowledge base of the subscriber,
- communicates with the subscriber agent through an agent communication language,
- delivers it a message (a complex event data, a secret password, a negotiation for an e-commerce or etc.) or requests something to be carried out (to update its database according to incoming data with the agvent, run a routine for updating its software, or etc.) according to its creation goal.

3.2. Subscription model:

Subscription model defines the selection capabilities of the publish/subscribe service. In designing a subscription model, the following properties should be considered in detail.

- *scope:* defines what parts of a publication can be evaluated and selected within subscriptions.
- *language power:* characterizes the language that defines subscription in terms of its expressive power.
- *language style:* declarative or imperative
- *other features:* extensibility (for example, by means of plug-ins), useful special operators such as a "certificate-based authentication" predicate that would select all the publication that a client can successfully authenticate.

The expressiveness of the subscription model is crucial for both the flexibility and the scalability of a notification service. Insufficient expressiveness can lead to unnecessary broad subscriptions stressing the network and raising the need for additional consumer-side filtering. On the other hand, scalable implementations of more expressive description models require complex delivery strategies [19].

<Rule_def> ::= <Rule>| <Rule> **where** <Condition>

<Rule> ::= **rule** identifier
onAgvent <Agvents>

<Agvents> ::= class/interface_type identifier |
class/interface_type identifier, <Events>

<Condition>::=Condition <Boolean_Operator> Condition
| (Condition) |! Condition
| <Exp> <Relation_Operator> <Exp>
| true | false

<Exp> ::= (<Exp>) | identifier
| <Exp> Arith_Operator <Exp>

<Arith_Operator> ::= + | - | * | /

<Relation_Operator>::= > | < | >= | <= | == |!=

<Boolean_Operator>::= and | or

Figure 3. The grammar of RDL in BNF notation

In the Agvent system, we use a rule based subscription model [3, 20], which uses the Rule Definition Language, whose grammar is shown in Figure 3. A rule is an expression or function that is evaluated or executed depending on the arrival of an agvent. It also defines necessary subscription information and the filtering conditions of the subscribers.

A rule definition is composed of three parts, each introduced by the keywords *rule*, *onAgvent* and *where*,

respectively. The first part sets a unique identifier for the rule, the second part specifies the type of the target agvent and the last part describes the conditions on which a filtered agvent should be caught. Some samples of subscription messages is shown in Figure 4.

rule rule_1 onAgvent Agv1 a. Simple Subscription	rule rule_2 onAgvent Agv2 where (price > 250 and price < 370) b. Filtered Subscription

Figure 4. Subscription Rules

Subscription can be classified in two groups:

- Simple Subscription is used to subscribe on an agvent type (as shown in Figure 4.a).
- Filtered Subscription is used to define a subscription with different criteria related to its attributes (as shown in Figure 4.b).

Subscribers are likely to select very specific information out of a varied information space. The increase in expressiveness through filters reduces the delivery of uninteresting notifications or even avoids totally.

3.3 Message/Agent flow in Agvent System

Communication between participants of the system (event servers, publishers and subscribers) is carried out using Java RMI. Due to architecture neutrality, Java and RMI handle geographically distributed heterogeneous machines and provide a transparent view to the participants. Figure 5 depicts the message/agent flow between components of the system and the details of the transfer are described below. Firstly, a publisher advertises its agvent type to the system. This advertisement is dispatched to all event servers in the dispatch service through a broadcast message.

1. If a subscriber is connected to an event server, it can get the advertisement list which includes a list of agvent types available on the system. All event servers have the same advertisement list.
2. If a subscriber decides to subscribe on an agvent type, it sends a rule based subscription message to the system to register itself and this message is dispatched to all event servers in the dispatch service (broadcast). These subscriptions are stored in subscription tables of knowledge bases, in each event server.
3. When a publisher observes and event and sends an agvent to the system, the event server to which it is connected receives the mobile agent and activates it. The agvent executes its pre-specified code to select the targets (subscribers or other event servers) from the subscription table.
4. Next, the agvent creates its clones and sends these to the targets.
5. A subscribers continues to receive published agvents until it unsubscribes from that particular agvent.
6. When a publisher stops publishing a certain type of agvent, it informs the system through an unadvertise message.

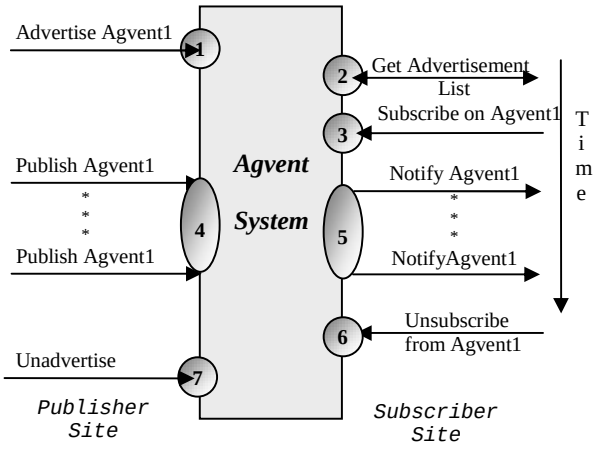

Figure 5. Message/agent flow in Agvent System

4. CONCLUSIONS

This paper presents a new model for agent based distributed events systems, the Agvent system, which combines the advantages of publish/subscribe communication and mobile agents into a flexible and extensible distributed execution environment. The system enables the utilization of mobile agents as mediators between participants of the system. By using mobile agent technology, routing operations and the control of the dispatching operation is controlled by Agvent messages. The publish/subscribe protocol allows for adding participants dynamically which extend the adaptability of the system. Currently a prototype of system is being implemented in Java.

REFERENCES

[1] T. O. Group. DCE 1.1: Remote Procedure Call. Technical Standard C706. The Open Group, Cambridge, MA, USA, 1997.

[2] Sun Microsystems Inc. Java RMI

[3] O. K. Sahingoz, N. Erdogan, “RUBCES: Rule Based Composite Event System”, in proceedings of XII. Turkish Artificial Intelligence and Neural Network Symposium (TAINN), Turkey, July (2003)

[4] O.K. Sahingoz, and N. Erdogan, (2003). "A Two-Leveled Mobile Agent System for Electronic Commerce", the journal of Aeronautics and Space Technologies Institute (ASTIN), 21-32.

[5] C. Liebig, B. Boesling, and A. Buchmann. A notification service for next-generation it systems in air

traffic control. In GI-Workshop: Multicast-Protokolle und Anwendungen, Braunschweig, Germany, May 1999.

[6] C. Bornhövd, M. Cilia, C. Liebig, and A. Buchmann. An infrastructure for meta-auctions. In Second International Workshop on Advance Issues of E-Commerce and Web-based Information Systems (WECWIS'00), San Jose, California, June 2000.

[7] N. Skarmeas and K. Clark. Content-based routing as the basis for intra-agent communication. In J. Müller, M. P. Singh, and A. S. Rao, editors, Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98), volume 1555 of LNAI, pages 345{362, Berlin, July 04{07 1999. Springer.

[8] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Transactions on Software Engineering, 27(9), 2001.

[9] M. Langheinrich, F. Mattern, K. Römer, and H. Vogt. First steps towards an event-based infrastructure for smart things. In Ubiquitous Computing Workshop (PACT 2000), Philadelphia, PA, USA, Oct. 2000.

[10] Object Management Group, "CORBAservices: Common Object Service Specification", Technical Report, Object Management Group, July (1998).

[11] "Java AWT: Delegation Event Model". Available online at http://java.sun.com/j2se/1.4.1/docs/guide/awt/1.3/designspec/events.html (2003)

[12] B. Krishnamurthy and D. S. Rosenblum, "Yeast: A General Purpose Event-Action System", IEEE Transactions on Software Engineering, 21(10):845–857, Oct. (1995).

[13] A. Carzaniga, "Architectures for an Event Notification Service Scalable to Wide-area Networks", PhD Thesis, Politecnico di Milano, Italy, December (1998).

[14] P.T.Eusgter, "TypeBased Publish/Subscribe", PhD Thesis. Ecole Polytechnique Federale De Lausanne, France, (2001)

[15] Th. Eugster Felber. The many faces of publish/subscribe. Technical report, Swiss Federal Institute of Technology in Lausanne (EPFL), 2001.

[16] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2):115–152, 1995.

[17] Jennings, N., R.: An agent-based approach for building complex software-systems. Communication of the ACM, Vol. 44, No. 4, acm Press, New York (2001)

[18] D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. Communications of the ACM, 42(3):88-91, March 1999.

[19] A.Carzaniga, D.S.Rosenblum,and A.L.Wolf.Design and evaluation of a wide-area event notification service.ACM Transactions on Computer Systems 19(3):332 –383,2001.

[20] O. K. Sahingoz, N. Erdogan, "RUBDES: Rule Based Distributed Event System", ISCIS XVIII - Eighteenth International Symposium on Computer and Information Sciences, LNCS, Springer-Verlag, 282-289, Nov. (2003).