AGVENT: AGENT BASED DISTRIBUTED EVENT SYSTEM

Ozgur Koray SAHINGOZ[1], Nadia ERDOGAN[2]

[1] Air Force Academy, Computer Engineering Department, Yesilyurt, Istanbul, TURKEY,
o.sahingoz@hho.edu.tr
[2] Istanbul Technical University, Electrical-Electronics Engineering Faculty,
Computer Engineering Department, Ayazaga, 80626, Istanbul, TURKEY
erdogan@cs.itu.edu.tr

Abstract. In recent years, a growing attention has been paid to the publish/subscribe communication paradigm as a means for disseminating information, also called events, through distributed systems on wide-area networks. As it allows events to be propagated in a way that is completely hidden to the component that has generated them as well as to its receivers, it is particularly interesting when easy reconfiguration and decoupling among components in a distributed system is required. 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 mobile agents as mediators between participants of an event based distributed system.

1 Introduction

In the traditional client/server computing model, which is used in RPC and RMI, communication is typically synchronous, tightly coupled and point-to-point. Clients invoke a method on the remote server and wait for the response to return. This type of communication requires clients and servers to have some prior knowledge of each other. 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. 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 [1].

In this paper, we present an agent based distributed event system, the Agvent (*Ag*ent e*vent*) 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. In the next section, we present a classification of publish/subscribe systems with references to related work. Section 3 introduces the design decisions and framework of Agvent system. Our conclusions are presented in Section 4.

2 Publish/Subscribe systems

Publish/subscribe programming paradigm is characterized by the complete decoupling of producers (publishers) and consumers (subscribers). The event service that provides message transfers between publishers and subscribers can be decomposed along three dimensions, time decoupling, space decoupling, flow decoupling. Publish/subscribe systems can be classified into four groups according to their subscription mechanism. Each one is discussed in detail below, with references to representative work.

Channel-based subscriptions: 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. The abstraction of the channel resembles to the one given by a mailing list. A user sends an e-mail to an address, and the message is forwarded to those who have registered to that mailing list. CORBA Event Service [2] adopts a channel-based architecture. Another widely used channel based model is the Java Delegation Event Model [3], which encapsulates events from the platform's Graphical User Interface.

Subject-based subscription: Some systems extend the concept of a channel with a more flexible addressing mechanism that is often referred to as *subject-based* addressing. In this case, an event notification consists of two different parts: a well-known attribute, *the subject* that 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 [4] 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

Content-based subscription: By extending the domain of filters to the whole content of notifications, some researchers obtain another class of subscriptions called content-based [6]. 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 gets more freedom in encoding the data upon which filters can be applied and that the event service can use for setting up routing information. Examples of event systems that provide this kind of subscription are Yeast [5] (uses a centralized structure) and SIENA [6]. 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 names. 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)”.

Type-based subscription: Type based publish/subscribe model [7], 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.

The main problem in large-scale publish/subscribe systems is routing notifications from producers to interested consumers. In [8], Gero Mühl tests different routing algorithms, such as flooding, simple routing, identity based routing, covering-based routing and merging-based routing, in its large scale publish-subscribe system. Both in this and previously described publish/subscribe systems, events are designed as simple event messages and dispatching these messages is the duty of event servers. A novel feature of the new model we propose for a distributed event system is that, it defines an event as a first class citizen of the system and gives it the autonomy and mobility to select and travel between system components.

3. AGVENT SYSTEM

The Agvent System is an agent based distributed event system whose framework is shown in Figure 1. 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 the publish/subscribe communication paradigm, in order to benefit the advantages of both. The advantages derived from the general characteristics of the publish/subscribe paradigm are the following [9].

- *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 inflexibility 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.

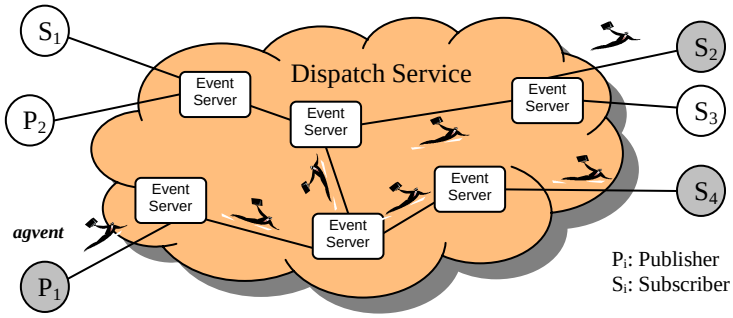

Fig. 1. Framework of the Agvent System

We want to add the general properties of software agents to this system [10].

- *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.

The system applies the capabilities of agents stated above to real-world “entities” on whose behalf they operate, whether that entity is 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. It 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, 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 represented by mobile agents that have their own goals, beliefs and behaviors that are loaded to agvents when they are created. When an agvent reaches to an event server, it examines the routing table of the server and selects its targets autonomously. This approach reduces the load and complexity of event servers a great deal.
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, an event server can, actually has to, access the content of the published event data before it can dispatch the 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 selected list. Therefore, the event server has no access to the content of the published event data, which simplifies its role and consequently facilitates the server development process. Information hiding also meets requirements of certain applications where confidentiality of event data is essential.
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 registers it on 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 [11, 12] 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.

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: either 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 [14].

In the Agvent System, we follow the rule based subscription model [1, 13], which uses the Rule Definition Language (RDL), whose grammar is shown in Figure 2.a. In this model, 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.

<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

a. The grammar of RDL in BNF notation

```
rule rule_1
onAgvent Agv1
```

b. Simple Subscription

```
rule rule_2
onAgvent Agv2
where (price > 250 and
        price < 370 )
```

c. Filtered Subscription

Fig. 2. Rule Based Subscription Model

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 are shown in Figure 2.b and Figure 2.c.

Subscriptions can be classified into two groups:

- Simple Subscription is used to subscribe on an agvent type (as shown in Fig.2.a).
- Filtered Subscription is used to define a subscription with different criteria related to its attributes (as shown in Fig 2.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 3 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 via a broadcast message.

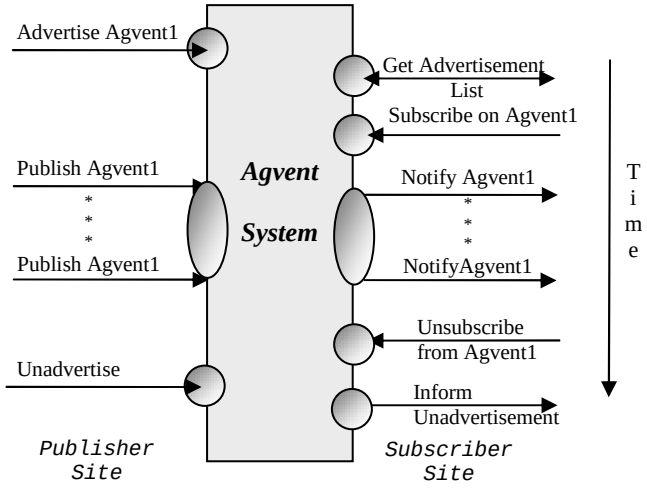

Fig. 3. Message/agent flow in Agvent System

1. If a subscriber is connected to an event server, it can obtain the advertisement list which includes a list of agvent types currently available on the system. All event servers hold the same advertisement list.
2. If a subscriber decides to subscribe on an agvent type, it sends a rule based subscription message to the event server it is connected to in order to have itself registered. Next, this message is broadcast to all event servers in the dispatch service. The details of the subscription are stored in subscription tables of knowledge bases in each event server.
3. Whenever a publisher observes an event, it creates and sends an agvent to the event server. When the agvent arrives on the event server, it starts to execute its pre-specified code to select its targets (neighbor event servers and/or registered subscribers) according to the information present in the subscription table and routing table.
4. Next, the agvent creates its clones and sends each one to a different target on the list, and after completing this task, it disposes itself.
5. A subscriber continues to receive published agvents until it issues an unsubscribe request for that particular agvent type. When an agvent arrives on a subscriber, it starts to communicate with the subscriber agent that is responsible for all subscriber operations, via an agent communication language (ACL). It

delivers the subscriber agent a message, which could either contain a complex event data, such as a secret password, a negotiation for an e-commerce or etc., or a request for an action to be carried out, such as updating its database according to incoming data carried by the agvent, running an update routine for its software, or etc., according to the execution logic of the current application.

6. When a publisher stops publishing a certain type of agvent, it informs the system through an unadvertise message. In this case, if no other publishers of that agvent are present, the system sends out a message to subscribers registered on that agvent type, informing them of the new situation, and adds these subscribers to a waiting advertisement list (WAL) created for that agvent. If, later, an advertisement message is received for that particular type of agvent, subscribers on its waiting advertisement list are alerted to subscribe in again, if they are still interested in that type of agvent.

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 major novelty of the model is that an event is represented by a mobile agent, an ***agvent***, which is treated as a first class citizen of the system and given autonomy and mobility features to select and travel between system components. The proposed model supports a rule-based subscription mechanism using a new language, RDL. The benefits of the proposed model are reduced network load, higher adaptability by allowing dynamic changes in system configuration, information hiding, asynchronous communication and flexibility of agent based execution. We think the new model will serve as an effective choice for several information-oriented applications, such as e-commerce or information retrieval, for its benefits stated above. Currently a prototype of system is being implemented in Java.

References

1. O. K. Sahingoz, N. Erdogan, “RUBCES: Rule Based Composite Event System”, in proceedings of XII. Turkish Artificial Intelligence and Neural Network Symposium (TAINN), Turkey, 2003
2. Object Management Group, “CORBAservices: Common Object Service Specification”, Technical Report, Object Management Group, 1998.
3. “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
4. G. Cugola, E. Di Nitto, and A. Fuggetta, “The JEDI event-based infrastructure and its application to the development of the OPSS WFMS”, Technical Report, CEFRIEL - Politecnico di Milano, Italy, 1998.
5. B. Krishnamurthy and D. S. Rosenblum, “Yeast: A General Purpose Event-Action System”, IEEE Transactions on Software Engineering, 21(10):845–857, 1995.

6. A. Carzaniga, “Architectures for an Event Notification Service Scalable to Wide-area Networks”, PhD Thesis, Politecnico di Milano, Italy, 1998.
7. P.T.Eusgter, “TypeBased Publish/Subscribe”, PhD Thesis. Ecole Polytechnique Federale De Lausanne, France, 2001
8. Gero Mühl, “Large-scale content-based publish/subscribe systems”, PhD Thesis, Darmstadt University of Technology, 2002.
9. Th. Eugster Felber. The many faces of publish/subscribe. Technical report, Swiss Federal Institute of Technology in Lausanne (EPFL), 2001.
10. M. Wooldridge and N. R. Jennings. “Intelligent agents: Theory and practice”, The Knowledge Engineering Review, 10(2):115–152, 1995.
11. Jennings, N. R., “An agent-based approach for building complex software-systems”, Communication of the ACM, 44(4), ACM Press, New York (2001)
12. D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. Communications of the ACM, 42(3):88-91, 1999.
13. 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, 2003.
14. 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.