An Agent Based Distributed Event System Framework

Ozgur Koray SAHINGOZ*, Nadia ERDOGAN **

*** Department of Computer Science, Air Force Academy,**

Yesilyurt, Istanbul, Turkey.

**** Electrical-Electronics Faculty, Computer Engineering Department,**

Istanbul Technical University, Ayazaga, 34469, Istanbul, Turkey.

sahingoz@hho.edu.tr erdogan@cs.itu.edu.tr

ABSTRACT

In the last years, event-based communication paradigm has been extensively studied and it is considered a promising approach to develop the communication infrastructure of distributed systems. In most of the event systems developed previously, events are defined as simple messages such as records, tuples or simple objects. Our work involves a new agent based distributed event system (Agvent System), in which events are represented by mobile and intelligent agents that are called **agvents** (***ag****ent e****vent***). In this paper, we present the framework of the Agvent System, and describe the communication protocol between system participants and the message/ agvent flow between system components.

Keywords : Distributed Event System, Mobile Agent, Publish-subscribe paradigm.

1. Introduction

In Internet-wide and ubiquitous systems, scalability is vital and must be ensured at all layers. In these systems, a messaging middleware may potentially have millions of dynamic clients and, therefore, must itself be implemented in a distributed fashion [1]. Event-based communication has become a new paradigm for building large-scale distributed systems of this type [2]. It has the advantages of loosely coupling communication partners, being extremely scalable, and providing a simple application programming model. In event-based systems, events are the basic communication mechanism and an event can be considered as a notification that something of interest has occurred within the system. Event-based communication generally implements what is commonly known as the publish/subscribe protocol.

The publish-subscribe communication paradigm has been recognized as a functional model particularly for distributed information systems, as it supports distributed and anonymous communication among several processes [3]. Participants have no explicit knowledge of each other and are only required to agree on the format of the data being exchanged. Participants can be classified in two categories: publishers, which produce data as a sequence of packet notifications, and subscribers, which inform about relevant notifications by either expressing a data subject or a condition on its content. It is the responsibility of publish-subscribe middleware to ensure delivery of published information to all interested subscribers.

The benefits of publish/subscribe protocol make it preferable 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]. Implementation of publish/subscribe techniques has also been described in the areas of mobile agents [7], software systems [1], and process control systems [2].

Clearly, a publish/subscribe system that relies on a centralized broker cannot be scalable. It may match an incoming event message against a large set of subscriptions very fast, but it will not be able to communicate with millions of clients. Moreover, a centralized broker is a single point of failure. In consequence, an implementation is needed that distributes the functionality of the service. The key for a scalable publish/subscribe system is using a dispatch service,

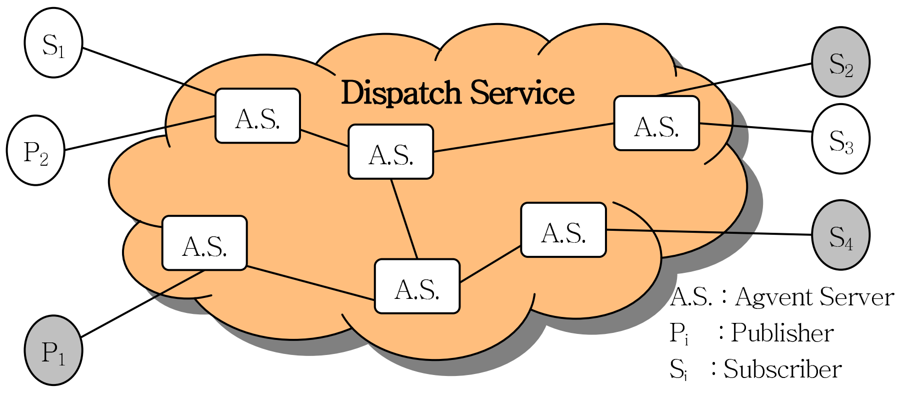

Figure 1. Framework of the Agvent System

which consists of cooperative event servers in a distributed topology.

Its data/filter model plays an important role in the design of a distributed event system. The data model defines how the content of notifications is structured, while the filter model defines how subscriptions are specified, i.e., how notifications are selected by applying filters that evaluate predicates over the content of notifications. Consequently, the filter model always depends on the underlying data model and there can be more than one filter model for a given data model.

In early distributed event systems, event data is represented by 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 serializable event objects. Rebeca [3], Siena [8], Gryphon [9], JMS [10], and the Corba Notification Service [11] are representative systems which define events as low level messages. Only type based publish/subscribe [12] system defines events as objects but it also doesn't provide them with autonomy. Our work proposes a new approach for distributed event systems through a model in which events are represented by mobile intelligent agents.

In this paper, we present an agent based distributed event system framework, the Agvent System framework, 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 the paper is organized as follows. In the next section, we present the characteristics of the Agvent System. Section 3 is described in detail the system framework. Section 4 explains the message/agvent flow and finally our conclusions and directions for future work are presented in Section 5.

2. Agvent System

The Agvent System is an agent based distributed event system which is implemented through a framework shown in Figure 1. It combines two developing technologies, mobile agents and the publish/subscribe communication paradigm, in order to benefit the advantages of both.

The main problem in large-scale publish/subscribe systems is routing notifications from producers to interested consumers. In [3], different routing algorithms are tested and compared. In previously described publish/subscribe systems, events are represented as simple event messages and dispatching these messages is the duty of event servers. A novel feature of the new model we propose is that, it allows events to be represented by mobile and intelligent agents, namely, *agvents***.**

Subscribers determine agvent types they are interested in and describe them in a rule form as defined in [13] to be processed by the system.

Publishers publish agvents of pre-specified type that search the knowledge bases of agvent servers to select lists of subscribers, clone themselves and direct a clone to each selected target subscriber.

The Agvent System differs from other distributed event systems with its distinct characteristics that are described below.

- ***Autonomous Events***. 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 main component of the system. Nevertheless, they are not autonomous.

In the Agvent System, events are not viewed as simple messages. On the contrary, they are represented as mobile agents that have their own goals, beliefs and behaviors that they acquire at creation. When an agvent reaches to an agvent server, it examines the routing table of the server and selects its targets autonomously after certain processing. This approach reduces the load and complexity of agvent servers as well.

- ***Agvent Based Subscription:*** In most distributed event systems, subscribers register on a channel, on a specific topic or a specific content of an event message. 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, specifying certain constraints based on its advertised attributes and behaviors.

- ***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 agvent server[1], selects the registered subscribers, clones itself and sends each agent clone to a subscriber on the selected list. Therefore, the agvent 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.

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

  In the Agvent System, a publisher creates its own agvent type and declares its properties and behaviors through an advertisement message sent to the Dispatch Service Once an agvent type is announced on the Dispatch Service, subscribers can register on agvents of that type.

[1] An agvent cannot reach the knowledge base of the server directly. It gets necessary information over Server Manager.

3. Agvent System Framework

Within the Agvent System, as in all push-based publish-subscribe systems, information flows from publishers to subscribers according to the specific selection criteria expressed by individual subscribers. Subscribers express their interests by means of subscriptions, while publishers simply advertise and publish agvents. The dispatch service accepts subscriptions and publications, and relays publications to subscribers that declared matching subscriptions. A network of distributed agvent servers constitutes the dispatch service of the system framework (see Figure 1). Each instance of the other two main components of the framework, publishers and subscribers, is directly connected to an agvent server, to which it sends its subscriptions, advertisements or publications. Every agvent server processes incoming subscriptions and advertisements according to some protocol, possibly redistributing them to other adjacent/neighbor agvent servers. Publications, actually agvents, move themselves to adjacent agvent servers and/or subscribers in a similar manner.

3.1. Publisher Subsystem

Publishers decide on what events are observable, how to name or describe those events, how to actually observe the event, and then how to represent the event as a discrete entity that is an agvent. To publish an agvent to the system, a publisher has to initialize a Publisher Subsystem on its machine. Publishers previously acquire the address (URL) of the agvent server that they will communicate with.

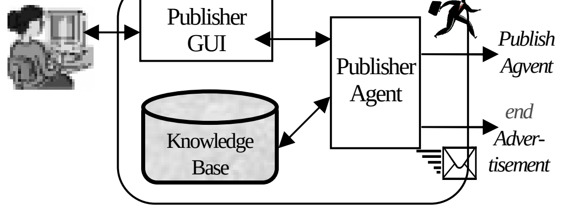

Figure 2. Publisher Subsystem Architecture

The Publisher Subsystem consists of three main components as shown in Figure 2.

1. *A Knowledge Base:* a database which contains information about past processes.
2. *A Graphical User Interface (GUI):* an interface used for human interaction.
3. *A Publisher Agent*: an agent which acts on behalf of the user.

Agvents may be created by two different sources:

- An agvent is either created automatically by the Publisher Agvent and sent to Dispatch Service, or
- The Graphical User Interface creates an agvent that meets the specifications of a predefined agvent type and sends it to Dispatch Service via the Publisher Agent.

Publisher Agent is the main processing unit of the Publisher Subsystem. It is a stationary agent that is created during the initialization step of the subsystem. It is responsible for advertisement and publication of agvents to the Dispatch Service.

3.2. Subscriber Subsystem

Subscribers determine agvent types they are interested in and describe them in a rule form which is processed by the Dispatch Service. A subscriber has to initialize a Subscriber Subsystem on its machine to join the system. In a manner similar to publishers, subscribers previously acquire the address (URL) of the agvent server that they will communicate with.
A subscriber registers itself to the system via a message that contains address and password information, and later subscribes on agvent types it is interested in, providing criteria to specify conditions on which it requests notification.

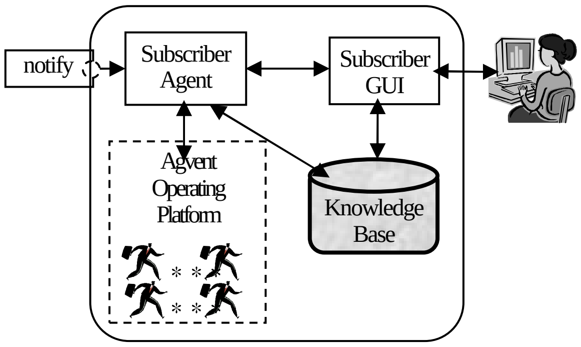

Figure 3. Subscriber Subsystem Architecture

The Subscriber Subsystem consists of four main components as shown in Figure 3.

1. *A Knowledge Base:* a database which contains information about past processes.
2. *A Graphical User Interface (GUI):* an interface used for human interaction.
3. *Subscriber Agent*: An agent that acts on behalf of the user.
4. *Agvent Operation Platform*: an execution platform that enables incoming agvents to run individually to reach their goals, such as carrying out update operations or delivering an important information.

The Subscriber Agent is a stationary agent, which is responsible for offering an interface to end users to enter information, control operations, or send subscription messages. The Subscriber Agent implements the Int_Subscriber interface depicted in Figure 4, which contains of two main methods which are used for receiving agvents (notify) and receiving Unadvertisement information from Dispatch Service.

```
public interface Int_Subscriber Remote
{
    public void notify(Agvent agv);
    public void inform (Advertisement adv);
}
```

Figure 4. Int_Subscriber Interface

3.3. Dispatch Service

A network of distributed Agvent Servers constitutes the Dispatch Service. The inner structure of an Agvent Server is depicted in Figure 5. It consists of six manager modules, which run concurrently in the system: Knowledge Base Manager, Subscription Manager, Publish Manager, Advertisement Manager, Agvent Dispatcher, and Server Manager. The task of each is explained below.

Knowledge Base Manager: This module coordinates the access and manipulation of four important tables which store data essential to system functioning. These are the following:

a. *Neighbor Table* keeps information (address, name…etc) of the adjacent agvent servers. This table is used especially for routing of subscription and advertisement messages.
b. *Advertisement Table* keeps information about the publishable agvents. It keeps not only public attributes, but also public behaviors of agvents with the information required as parameter. All agvent types

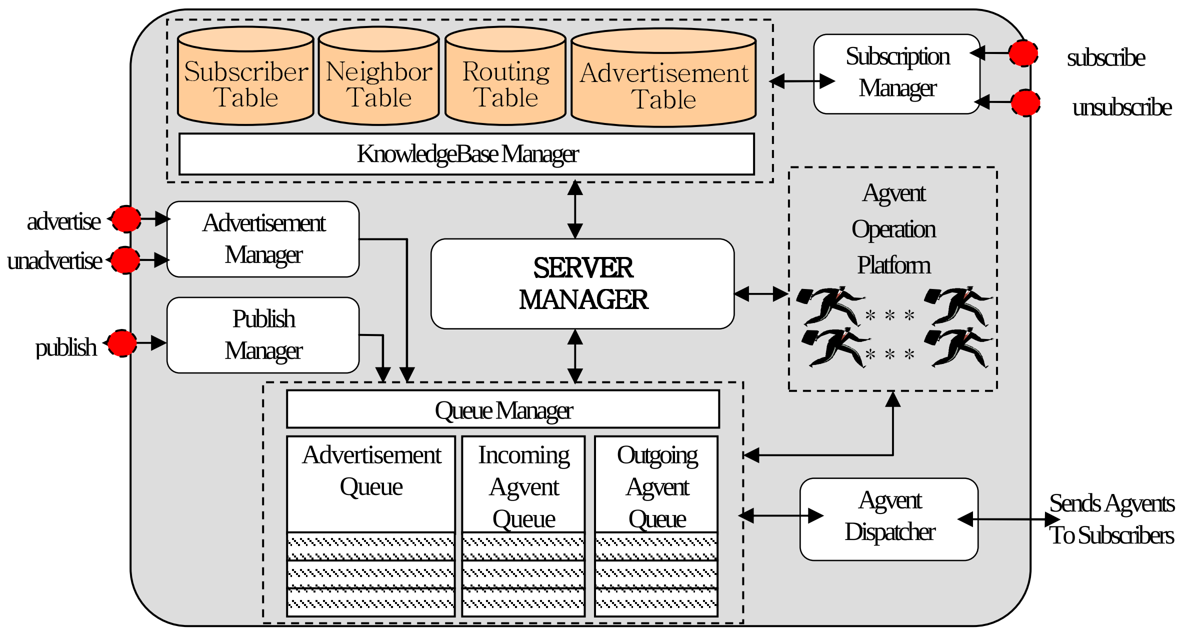

Figure 5. Inner structure of the Agvent Server

have to be advertised before they are published.

c. *Subscription Table* keeps subscriber related information, such as name, password and communication address, provided during system registration.

d. *Routing Table* is the most frequently accessed data store of the system. It keeps the subscription criteria of subscribers and used for routing of agvents.

The Queue Manager: This module coordinates the access and manipulation of three queues which are used to allow concurrent execution of manager modules.

a. *Advertisement Queue* keeps the incoming advertisement messages in a specific format.

b. *Incoming Agvent Queue* contains the serialized form of incoming agvents from publishers.

c. *Outgoing Agvent Queue* contains the serialized form of outgoing agvents from Agvent Servers.

Subscription Manager: This module receives subscription messages from subscribers via *subscribe* and *unsubscribe* methods, verifies the id and password information with the Subscriber Table, and proceeds with the necessary actions.

Publish Manager: This module receives the serialized forms of incoming Agvents from publishers via *publish* method and adds them to the Incoming Agvent Queue.

Advertisement Manager: This module receives the advertisements (also unadvertisements) via *advertise* method (or *unadvertise* method) and adds them to the Advertisement Queue.

Agvent Dispatcher: This module checks the Outgoing Agvent Queue and sends the serialized forms of the agvents to their target host.

Server Manager: This module is the main component of the Agvent Server as it has control over all operations in the system. It evaluates incoming advertisement requests. It is also responsible of the Agvent Operation Platform, which enables incoming agvents to execute individually and process data to fulfill their goals. Server Manager also mediates between agvents and Knowledgebase of the server for getting results of necessary inquiry operations.

```
public interface Int_AgventServer extends Remote
{
    public void publish(Agvent agv);

    public void subscribe(Subscription sub);

    public void unsubscribe(Subscription sub);

    public void advertise(Advertisement adv);

    public void unadvertise(Advertisement adv);

    public Advertisement[] getAdvertisementList()
}
```

Figure 6. Int_AgventServer Interface

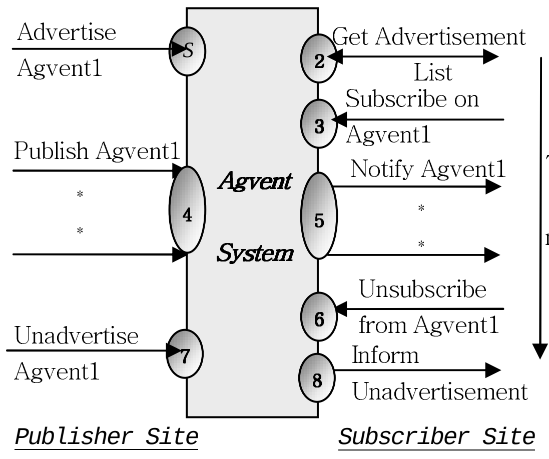

Figure 7. Execution Flow in the Agvent System

4. Message/Agvent Flow

The message and agvent flow in the dispatch service is shown in Figure 7. and is described below:

a. Firstly, a publisher announces its agvent type to the system. This advertisement is dispatched to all agvent servers in the dispatch service.

b. If a subscriber is connected to an agvent server, it can obtain the advertisement list/table which includes a list of agvent types currently available on the system. All agvent servers hold the same advertisement list/table.

c. If a subscriber decides to subscribe on an agvent type, it sends a rule based subscription message to the agvent server it is connected to in order to have itself registered. Next, this message is broadcast to all agvent servers in the dispatch service. The details of the subscription are stored in subscription tables in each agvent server.

d. Whenever a publisher observes an event, it creates and sends an agvent to the agvent server. When the agvent arrives on the agvent server, it starts to execute its pre-specified code to select its targets (neighbor agvent servers and/or registered subscribers) according to the information present in the subscription table and routing table.

e. Next, the agvent creates its clones and sending each to a different target, after which it disposes itself.

f. 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.

g. 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.

5. 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 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 is being implemented in Java.

References

[1] 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.
[2] 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)
[3] G. Mühl, "Large-scale content-based publish/subscribe systems", PhD Thesis, Darmstadt University of Technology, 2002.
[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. GI-Workshop: Multicast-Protokolle und Anwendungen, 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 1999. Springer.

[8] A. Carzaniga., Architect.for Event Notification Service Scalable to WAN. PhD thesis, Politec. di Milano, Italy, Dec. 1998.

[9] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC 1999), pages 53-61, USA, 1999.

[10] Sun Microsystems, Inc. Java Message Service Spec. 1.1, 2002.

[11] Object Management Group. Corba notification service. OMG Document telecom/99-07-01, 1999.

[12] P.T. Eusgter, “Type Based Publish/Subscribe”, PhD Thesis. Ecole Polytechnique Federale De Lausanne, France, 2001.

[13] O. K. Sahingoz, N. Erdogan, “RUBDES: Rule Based Distributed Event System”, ISCIS XVIII - Eighteenth International Symposium on Computer and Information Sciences, LNCS Vol. 2869, Springer-Verlag: 282-289, 2003.