Dispatching Mechanism of an Agent-Based Distributed Event System

Ozgur Koray Sahingoz[1] and Nadia Erdogan[2]

[1] Air Force Academy, Computer Engineering Department, Yesilyurt, Istanbul, Turkey, sahingoz@hho.edu.tr

[2] Istanbul Technical University, Electrical-Electronics Faculty, Computer Engineering Department, Ayazaga, 34469, Istanbul, Turkey erdogan@cs.itu.edu.tr

Abstract. In recent 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 previously developed event systems, events are defined as simple messages, such as records tuples, or simple objects. Our work introduces a new approach which allows events to be represented as mobile intelligent agents that are called **agvents** (***ag**ent e**vent***) in the context of an agent-based distributed event system, the Agvent System. In this paper, we present the dispatching mechanism implemented by the Agvent System, where agvents are responsible for determining their own paths through the network, selecting target nodes and moving themselves to these targets.

1 Introduction

Event-based communication model represents an emerging paradigm for middleware that asynchronously interconnects the components that comprise an application in a potentially distributed and heterogeneous environment [1], and has recently become widely used in application areas such as large-scale Internet services. Event-based communication model supports either one-to-many or many-to-many communication pattern that allows one or more application components to react to a change in the state of another application component. Event-based communication generally implements what is commonly known as the publish/subscribe protocol.

The publish/subscribe protocol is very well suited for connecting loosely coupled large-scale applications in the Internet. 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 the communicating parties.

As pointed out in [2], two open problems of distributed publish/subscribe systems (especially content-based systems) are security issues of the system and routing of events. Security is an important issue, because it is possible for a subscriber to be interested in an event (have a subscription for a particular event) but not be authorized to read that event (because of restrictions from a publisher). Although defining security architecture is important, it is outside the scope of this paper.

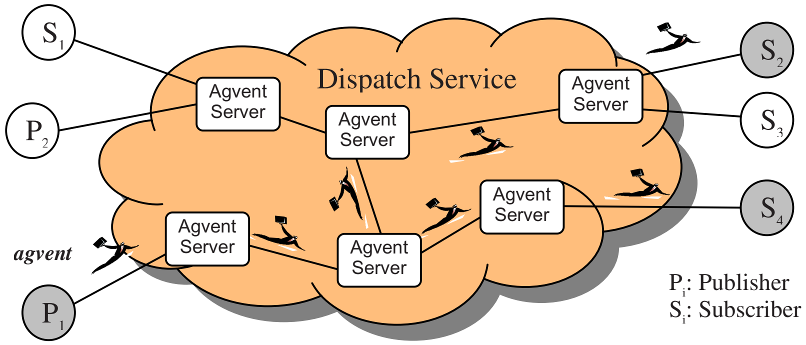


Fig. 1. Framework of Agvent System

For the scalability of distributed event systems, routing strategy is an important issue also. Generally a routing table is used in an event server to select the outgoing servers and clients, to which incoming messages will be sent. Different strategies are used for composing routing tables such as flooding, simple routing and content based routing [3]. In all these strategies, an event is defined as a simple message and it is dispatched to selected targets by a manager agent in the event server. In this paper, we propose a new approach for routing incoming events to targets. An important feature of our model is that, it allows events to be represented as mobile and intelligent agents that are called **agvents.** An agvent is an autonomous entity that acts on its own to reach its goal. A significant capability of an agvent is its ability to discover target nodes, namely subscribers, which need to be notified of the occurrence of an event, and to route itself to them. This is accomplished by enabling the published agvent itself search the knowledge base of an Agvent Server, select the registered subscribers, clone itself and send each agent clone to a subscriber on the selected list.

In this paper, we describe the dispatching mechanism of Agvent System, which provides high flexibility, scalability, and resilience to adverse network conditions. The goal of the agvent model is to keep the support required from agvent servers in the network to a minimum, placing intelligence in agvents rather than in individual nodes. This approach provides flexibility and obviates the need for the potentially impossible task of updating all nodes in a network for the implementation of a new application or protocol.

In the following, first Agvent System is described in Section 2. Next, the dispatching mechanism of Agvent System is explained in detail in Section 3, and finally, we present our conclusion and plans for future work in Section 4.

2 Agvent System

The framework depicted in Figure 1 represents the infrastructure of Agvent System [4] which is an agent based distributed event system that aims to meet the communication requirements of distributed and cooperative applications through the event-based design style. Agvent System combines two developing technologies, mobile agents and the publish/subscribe communication paradigm, in order to benefit the advantages of both. Its framework consists of three main components: Publishers, Subscribers and a Dispatch Service. `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, actually an agent that is called an *agvent*. `Subscribers` determine the particular agvent types they are interested in and describe them in the form of a rule which is processed by the Dispatch Service. *`Dispatch Service`*, which is responsible for dispatching incoming agvents to registered subscribers, is the main component of the Agvent System and is comprised of a network of distributed *Agvent Servers*. In a server topology which contains cycles, additional care must be taken to avoid cyclic routing. In order to simplify implementation issues, only *acyclic* topologies are considered in this project.

The entire Agvent system model is built under the assumption that the node/server architecture must be kept as simple and flexible as possible. Routing for instance, is agvent specific and not node/server specific. In this way, different agvents can execute different selection algorithms simultaneously on the same server. The Agvent Server, on the other hand, is responsible for storing subscription information in its knowledge base and provides an `Agvent Operation Platform` where agvents can migrate to and pursue their execution. The inner structure of an Agvent Server is depicted in Figure 2. Every Agvent Server processes incoming subscription and advertisement requests according to a protocol, which includes their propagation to adjacent/neighbor Agvent Servers. An Agvent Server communicates with publishers, subscribers and other Agvent Servers through the set of methods shown in Figure 2. Agvents move themselves to adjacent Agvent Servers and/or subscribers similarly to reach target nodes.

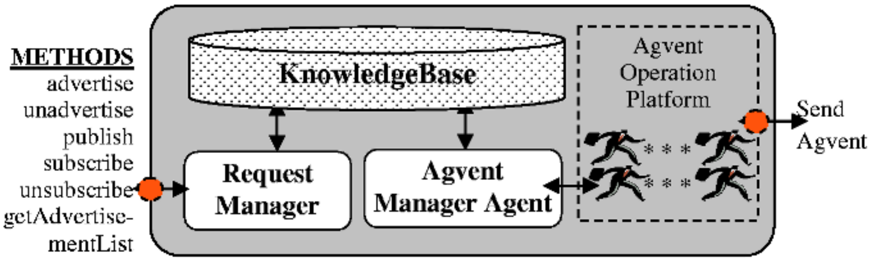


Fig. 2. Inner Structure of an Agvent Server

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

- **Autonomous Events:** 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. This approach adds to the flexibility of the system and reduces the load and complexity of agvent servers as well.
- **Self-Routing Agvents:** Agvents are self-routing, that is, they are responsible for determining their own paths through the network, utilizing a minimal set of facilities provided by Agvent Servers. Agvent Servers in the network support incoming agvents by providing a simple, architecturally independent environment for the receipt and execution of agvents.

- **Agvent Based Subscription:** Subscribers register on agvent types. For example, a subscriber can register on an agvent, which is an instance of "BookAgvent" class, specifying certain constraints based on its advertised attributes and behaviors.
- **Information Hiding:** The published agvent itself searches the knowledge base of the agvent server *by talking with the* `Agvent Manager Agent`, selects the registered subscribers, clones itself and sends each agent clone to a subscriber on the selected list. Therefore, an 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:** A publisher creates its own agvent type and declares its properties and behavior 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.

2.1 Agvents

An agvent is a collection of code and data that migrates through the network, routing itself at each node on its path towards its target node. Agvents are created by publishers and sent out to subscribers over the Dispatch Service which is comprised of a network of distributed nodes. Each node hosts an Agvent Server which provides an operation platform for agvent execution, where incoming agvents execute their code in order to achieve their prescribed objectives. An agvent carries its routing procedure and routes itself at each node on the path toward a node of interest. To perform routing, an agvent communicates with Agvent Manager Agent (AMA) (Figure 2), using FIPA Agent Communication Language [5] in order to obtain the required information from the knowledge base of the Agvent Server.

After receiving the required information from AMA, an agvent may clone itself to create new agvents to be sent out to other Agvent Servers or subscribers on the network. Thus, an agvent can eventually generate multiple agvents, even though it starts out as a single one.

```
class BookAgvent extends Agvent
        {       private String  Author;
                private String  Name;
                private float[] Dimensions ;
                private String  Publisher;
                private Date    Publish ;
                private int     ISBN;
                private float   ListPrice;
                private boolean ReferenceContains(String AuthorName) {…}
                private boolean TOCContains(String topic)  {…}
                privatefloat WholesalePrice(int amount,String destination) {…}
                .........
                ......... }
```

Fig. 3. Class definition of BookAgvent

The following is a sample scenario, an event which involves the introduction of a new product. A publishing house wishes to announce a new book to interested subscribers, according to their subscription criteria. It first creates the class for an agvent, namely `BookAgvent` (see Figure 3), which contains only the necessary information for subscribers, however restricting access to detailed information of the book from both servers and subscribers of the system. Next, the publishing house creates an agvent instance of `BookAgvent`, embeds the necessary information of this new book into this agvent, and publishes this agvent to the system.

2.2 Life Cycle of an Agvent

An agvent has a four-stage execution life cycle on different nodes of the Agvent System. Flow of transition between stages, depicted in Figure 4, is as the following: (1) being created on a publisher, (2) migrate to an agvent server/subscriber of interest, (3) execution upon a target subscriber and (4) disposal upon completion of mission.

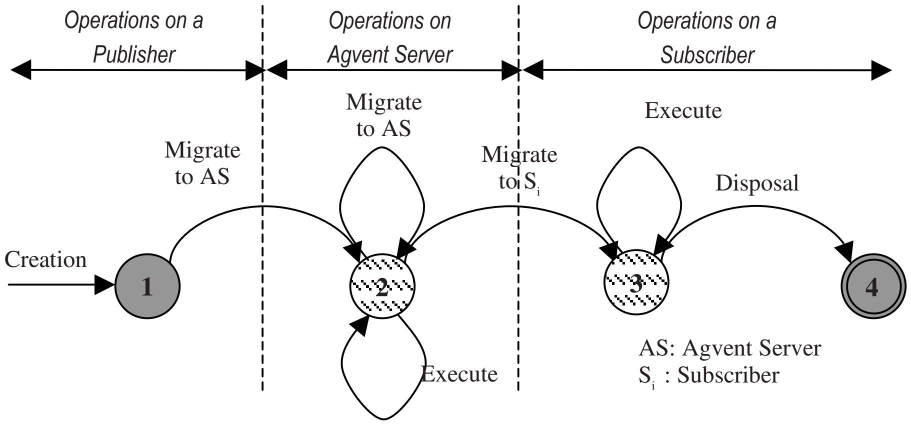


Fig. 4. Life Cycle of an Agvent

Publishers which decide on what events are observable and how to describe them as agvents. The life cycle of an agvent starts with its being created by a publisher on the observation of an event. Next, it routes itself to an Agvent Server where it identifies its target set of subscribers and a path which will take it to its destination nodes. At this point, new agvents may be spawned to be sent to different target nodes. After arrival on a target, actually a subscriber node, the agvent carries out a prespecified set of actions to reach its goals, and upon completion, disposes itself.

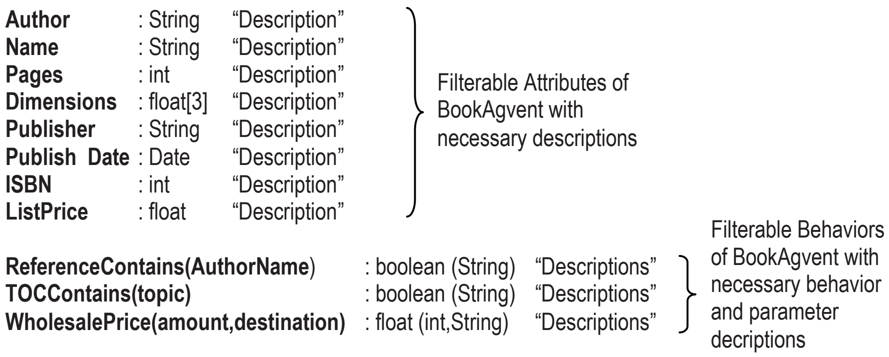
3 Dispatching Mechanism of Agvent System

The Agvent System implements three different dispatching mechanisms for different entities of the system, namely advertisements, subscriptions and agvents. To carry out the dispatching process correctly, each Agvent Server maintains a knowledge base to keep information to be used to route incoming request messages to local clients or neighboring Agvent Servers, the decision being based on the content of the message.

3.1 Dispatching of Advertisements

Every publisher knows of an Agvent Server to which it issues an advertisement request to advertise intent to publish a particular kind of agvent and make it visible to all subscribers of the system. An advertisement describes the properties of the relevant agvent, containing not only the attributes (with necessary descriptions) but also its behaviors that are to be exposed to subscribers. By getting this information, a subscriber can register on an agvent with constraints on these properties, as predicates asserted on the attributes or on the return values of behaviors on a list of input parameters prespecified by the subscriber.

Returning to the previous scenario, after developing the "BookAgvent" class, the publishing house issues an advertisement message for this agvent and sends it to the Dispatch Service. This message is structured as an instance of "*Advertisement*" class, which contains the necessary data structures to store information on filterable attributes and behaviors of the agvent, as depicted in Figure 5. The Dispatch Service distributes this message to all its constituent Agvent Servers through flooding and accepts subscriptions on this advertisement.



Author	: String	"Description"	Filterable Attributes of BookAgvent with necessary descriptions
Name	: String	"Description"	
Pages	: int	"Description"	
Dimensions	: float[3]	"Description"	
Publisher	: String	"Description"	
Publish Date	: Date	"Description"	
ISBN	: int	"Description"	
ListPrice	: float	"Description"	
ReferenceContains(AuthorName)	: boolean (String)	"Descriptions"	Filterable Behaviors of BookAgvent with necessary behavior and parameter decriptions
TOCContains(topic)	: boolean (String)	"Descriptions"	
WholesalePrice(amount,destination)	: float (int,String)	"Descriptions"	

Fig. 5. An Advertisement of a BookAgvent

When an Agvent Server receives an advertisement, it stores the content of the message in an *advertisement table* along with the addresses of both the Agvent Server from which it received the message and the publisher node, that is, the owner of the advertisement. Next, it forwards the message to adjacent Agvent Servers. Thus, with this approach, the advertisement message is propagated to all Agvent Servers on the Dispatch Service and a trail of backward pointers from Agvent Servers to the publisher node is created. Consequently, it becomes possible to reach the publisher of the advertisement from any Agvent Server when successive locations on this trail are visited. Subscriptions are also dispatched over these routes created during the advertisement process. When a subscriber asks for the whole advertisement list or enquires a specific type of advertisement, the Agvent Server replies with the requested information and waits for subscriptions. Advertisements remain in effect until they are cancelled by a call to unadvertise.

3.2 Dispatching of Subscriptions

Subscriptions express the interests of applications/subscribers. With a subscription, an application can instruct the Dispatch Service its request to receive a certain agvent type through a filtering function. Setting a filter with a subscription means defining a predicate that is stored in the Agvent Server and that will be evaluated by every incoming agvent of that type. It is of fundamental importance to define the domain of these operations. In other words, it is crucial to determine

- which attributes of an agvent are filterable for the evaluation of subscriptions
- what kinds of primitive predicates and connectors are available (such as ">, >=,<,<=,!,!=,=…etc").

Agvent system uses the *Subscription* class which contains the necessary data structures to describe constraints on both attributes and behaviors of an agvent. A sample subscription for the previously advertised BookAgvent may contain the constraints depicted in Figure 6.

Author =" Valentina Plekhanova"
Pages > 200
Publish Date > "January 1, 200"2
} Constraints on attributes of BookAgvent

ReferenceContains("Alonso, E.") == true
WholesalePrice(1000, "Istanbul"**) < $**150.000
} Constraints on behaviors of BookAgvent

Fig. 6. A Subscription sample on BookAgvent

Since subscriptions define the potential targets of agvents, they are used by the Dispatch Service to create a routing table which is used by agvents for self-routing. Subscriptions can be matched repeatedly until they are cancelled by an unsubscribe call. With this policy, an agvent selects a target node and moves itself there only if an interested subscriber resides on that node. However, such a policy requires every subscription to be propagated to every Agvent Server in the system.

When a subscription message reaches an Agvent Server, either from a subscriber client or from another Agvent Server, the server adds the new subscription information to its routing table in the knowledge base and propagates that subscription message to adjacent servers from which it has received the advertisement of the particular type of agvent the subscription is on. Every subscription is stored and forwarded from the originating Agvent Server to all other Agvent Servers in the network.

3.3 Dispatching of Agvents

An agvent is a collection of code and data that migrates through the network, routes itself at each node on the path, and executes on nodes of interest. An agvent determines its own path through the network, utilizing the minimal set of facilities provided by nodes. A key challenge in this model is the ability to discover target nodes, and to route itself to them. An agvent specifies target nodes with matching subscribers after applying filtering specifications. The next step requires the migration of clones of the agvent to each target. This execution cycle is completed in the following steps.

- First, an agvent has to be admitted at the destination node. A launcher task at an agvent server is to continuously receive agvents arriving from other servers or publishers.
- Second, upon acceptance, this agvent is activated on Agvent Operation Platform and scheduled for execution as a thread. During its execution the agvent may yield the processor and wait for data from Agvent Manager Agent and the routing behavior of an agvent chooses target nodes(agvent servers or subscribers). The execution performed at each step may differ based on peculiar properties of that node.
- Third, when the agvent completes its execution on the current node, it clones itself for transport to each of these target nodes. Agvent Manager Agent sends these clones to the destinations as dictated by agvent's behaviors.

4 Conclusion

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 as a mobile intelligent 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. Agvents have their own identity and behavior which permit them to actively navigate through the underlying dispatching system, and carry out various tasks at the nodes they visit. Agvents operate independently of the sending application, therefore removing any dependencies with the application. The 'intelligence' is in the agvents themselves rather than in the network. 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. Bacon, J., Moody K., Bates J., Hayton R., Ma C., Mcneil A., Seidel O., and Spiteri M. "Generic support for distributed ap-plications." IEEE Computer 33, 3 (2000), 68–76.
2. Carzaniga, A. and Wolf, A. "Content-based Networking: A New Communication Infrastructure." NSF Workshop on Infrastructure for Mobile -Wireless Systems. Oct., 2001.
3. Mühl, G. Large-scale content-based publish/subscribe systems. PhD thesis, Darmstadt University of Technology, September 2002.
4. Sahingoz, O. K. and Erdogan, N. "AGVENT: Agent Based Distributed Event System", accepted for presentation in 30th Conference on Current Trends in Theory and Practice of Computer Science, (SOFSEM-2004), Czech Republic, 2004
5. FIPA Agent Communication Language Specifications. 2000. HTML, `http://www.fipa.org/repository/aclspecs.html.`