Computing and Informatics, Vol. 26, 2007, 489–506

AGENT-BASED FAULT TOLERANT DISTRIBUTED EVENT SYSTEM

Ozgur Koray Sahingoz

Turkish Air Force Academy
Computer Engineering Department
Yesilyurt, 34149, Istanbul, Turkey
e-mail: sahingoz@hho.edu.tr

A. Coskun Sonmez

Yildiz Technical University
Computer Engineering Department
Yildiz, 34349, Istanbul, Turkey
e-mail: acsonmez@ce.yildiz.edu.tr

Revised manuscript received 20 February 2007

Abstract. In the last years, event-based communication style has been extensively studied and is considered a promising approach to develop large scale distributed systems. The historical development of event based systems has followed a line which has evolved from channel-based systems, to subject-based systems, next content-based systems and finally type-based systems which use objects as event messages. According to this historical development the next step should be usage of agents in event systems. In this paper, we propose a new model for *Agent Based Distributed Event Systems*, called *ABDES*, which combines the advantages of event-based communication and intelligent mobile agents into a flexible, extensible and fault tolerant distributed execution environment.

Keywords: Distributed event system, fault tolerance, agvent, agent event, advertisement based routing, publish/subscribe systems

1 INTRODUCTION

The standard models for client/server communication in distributed object computing are based on synchronous method invocations as used in COM+ [1], Java RMI [2], and CORBA [3]. Clients invoke a method on the remote server and wait for the response to return as depicted in Figure 1 a). This approach has several limitations like *tightly coupled, synchronous and point to point* communication. This type of communication exhibits critical scalability problems especially in data-centric environments [4].

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 communication protocol serves these needs by its three decoupling characteristics as shown in Figure 1 b) [5]:

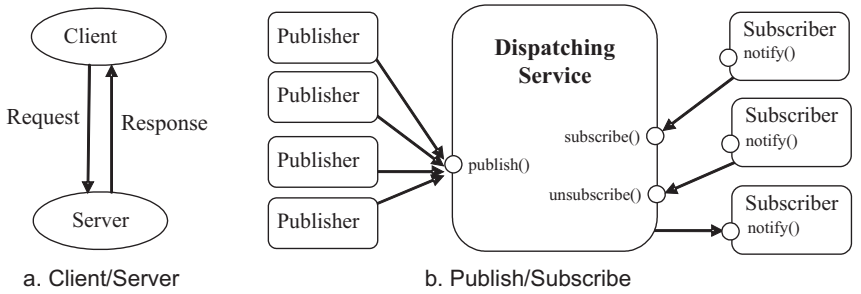

Fig. 1. Client/server and publish/subscribe communication models

Space decoupling: Publishers which are the producers of events do not need to address subscribers which are consumers of events and vice versa. Instead, subscribers simply specify the notifications they are interested in. This loosely coupled approach facilitates flexibility and extensibility because new subscribers and publishers can be added, moved or removed easily.

Flow decoupling: Communication is asynchronous, thereby removing the disadvantages and inflexibility of synchronous communication.

Time decoupling: Publishers and subscribers 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.

The publish/subscribe communication paradigm has been recognized as a functional model particularly for *Distributed Event Systems* in which events are the basic communication mechanism. An event can be seen as a notification (data structure) that something of interest has occurred within the system. Components either act as event sources and publish new events, or act as event consumers and

subscribe to events by providing a specification of events that are of interest to them.

Event systems consist of distributed components which communicate through the exchange of event messages which can be defined as simple messages, such as records, tuples or simple objects. Middleware using an event based communication model is appropriate to address the requirements of distributed applications for large scale and heterogeneous environments which require a less tightly coupled communication relationship between their components. The benefits of event based systems make them preferable for implementing information-driven applications. For example, they are well suited for information dissemination applications like news delivery, stock quoting, digital libraries [6], e-commerce [7, 8], air traffic control [9] and dissemination of auction bids [10]. However, there are also some application independent implementations of event systems like [11, 12, 13, 14].

Mobile agents are a new and fascinating design paradigm and aims to introduce the required intelligence into distributed information processing. Traditionally designed distributed systems have some problems and might have to handle workload, the trend to open large numbers of customers, direct access to services/goods and user mobility [15]. Intelligent mobile agents overcome most of these problems by their characteristics which are described by Wooldridge and Jennings as follows [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 agents and humans in order to complete their own problem solving and to help others with their activities.

Reactivity: Agents should perceive their environment 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.

We intend to combine advantages of both publish-subscribe communication protocol and intelligent mobile agents in a scalable and fault tolerant distributed event system. In this paper, we present an Agent Based fault tolerant Distributed Event System (ABDES), which exploits mobile agents as mediators between publishers and subscribers. The rest of this paper is organized as follows: In the next section, we present a classification of distributed event systems with references to related work. Section 3 describes properties of ABDES. Message and agent flow between system components is described in Section 4 and Section 5 explains the dispatching mechanism of the system in detail. Section 6 presents the fault tolerance mechanism of ABDES and finally Section 7 concludes the paper.

2 RELATED WORKS

A distributed event system consists of allowing some components called subscribers to express their interests in some kind of information, while allowing other components called publishers to publish this information. The dispatching system is responsible for matching publications to subscriptions and forwarding them to interested subscribers according to their subscription messages. Subscriptions define the potential targets of event messages and can be used by the dispatching system to route of event messages. Therefore the subscription mechanism is one of the crucial parts of the system. Distributed event systems can be classified into four groups according to their subscription mechanism. According to historical development these mechanisms are discussed in detail below.

2.1 Channel-Based Subscription

The simplest subscription mechanism is what is commonly referred to as a channel. Some systems like CORBA Event Service [3] and Java Delegation Event Model [17] implement this subscription mechanism. In these systems subscribers register and 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 subscribers listening that channel. Channel based systems have some inherent disadvantages. The expressiveness and the filtering capability of channels is rather limited because notifications can only be classified with respect to a number of channels.

2.2 Subject-Based Subscription

Some systems like Jedi [18] and SwiftMQ [19] 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. Subject based systems provide more powerful notification selection than channels. Nevertheless, subjects have a number of drawbacks. They still have a limited expressiveness because subjects are only suitable to divide the event dispatching space with respect to one dimension.

2.3 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 subscriptions

which 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. Because of its expressive form and filterable design of event messages most of distributed event systems like Rebeca [7], Gryphon [8], Elvin [11], Lesubscribe [12], Siena [14] and Dadi [20] implements this type of subscription mechanism. Content-based systems are contrasted with channel-based and subject-based systems, because the selection is done based on the whole content. The other strategies offer only a set of well-defined attributes for selection purposes. Matching event messages to the subscriptions brings a burden to the underlying system which influences the scalability.

2.4 Type-Based Subscription

Type-based subscription model is a new model of subscription that has been developed to access event data in a more structured manner by using the features of object oriented programming languages. The original motivation for introducing type-based systems was its supports on type safety and encapsulation which are important factors in producing robust, error-free and stand-alone applications. In previous subscription models 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. In this subscription model event messages are defined as objects, which are strongly typed as in most object oriented languages. Subscribers register on type of event messages. JECho [21], Hermes [6, 22] and type-based publish subscribe system [23] use this type subscription mechanism. In the type-based systems events are filtered according to their type. This is enabled by a close integration of the programming language and the middleware to allow compile-time type checking.

According to this historical development described above, the next step should be usage of agents in event systems. In previously mentioned 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. Our work proposes a new approach for distributed event systems through a model in which events are represented by intelligent mobile agents.

3 AGENT BASED DISTRIBUTED EVENT SYSTEM

The historical development of publish/subscribe systems has followed a line which has evolved from channel-based systems, to subject-based systems, next to content-based systems and finally to type based systems. We thought the next step should contain agents in the system and developed an *Agent Based Distributed Event System-ABDES* [24] which defines events (called agvents-*AGent eVENTs*) as first class members of the system.

ABDES 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 agent and given autonomy and mobility features to select and travel between system components. Using events as intelligent mobile agent potentially leads to higher type safety and better encapsulation of events than traditional publish/subscribe communication styles, and enables the programmer to focus on agent oriented abstractions and modeling in contrast to lower level details.

The benefits of the 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, information retrieval, publication dispatch systems, distributed software and virus definitions updating. The ABDES system consists of four main components as shown in Figure 2:

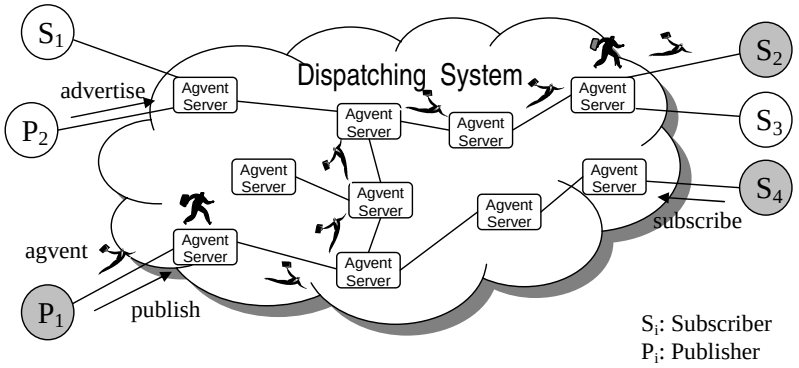

Fig. 2. Framework of the ABDES system

Publishers decide what events are observable, how to name or describe those events, how to actually observe an event and then how to represent an event as a discrete entity that is an agvent.

Subscribers determine agvent types they are interested in and describe them in a subscription form which is processed by the Dispatch Service.

Agvent Servers are responsible for storing advertisements and subscriptions information in its knowledge base and provides an Agvent Operation Platform where agvents can migrate to and pursue their execution.

An agvent is a collection of code and data that migrates through ABDES. A significant capability of an agvent is its ability to discover target nodes, namely subscribers which demanded to be notified of the occurrence of an event, and to route itself to these subscribers. This is accomplished by enabling the published

agvent to 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. Each agvent server provides an operation platform for agvent execution, where incoming agvents execute their code in order to achieve their prescribed objectives. An agvent determines its own path through the network, utilizing the minimal set of facilities provided by agvent servers.

Within ABDES, as in all push-based publish-subscribe systems, information flows from publishers to subscribers over dispatching system according to the specific selection criteria expressed by individual subscribers. Dispatching system consists of cooperative agvent servers in a distributed topology. 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. The ABDES system differs from other distributed event systems by its distinctive 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 ABDES system, events are not viewed as simple messages. On the contrary, they are represented as intelligent mobile agents which have their own goals, beliefs and behaviors that they acquire at creation. When an agvent reaches an agvent server, it examines the knowledge base 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 on a specific content of an event message. In the ABDES system, subscribers register on agvent types and define their filter not only on agvents' attributes but also on their behaviors. For example, a subscriber can register on an agvent, which is an instance of 'Agvtype1' 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 ABDES system, the published agvent itself searches the knowledge base of the agvent server, 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 ABDES 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.

Self routing agvents: In previously developed event systems, routing of events is the main duty of the event servers. Because of the intelligent mobile agent structure of agvents, route of an agvent is determined by itself with minimum help of agvent servers. Role of an agvent server is providing an execution platform for agvents and storing advertisement/subscription messages for necessary routing operations.

Double side filtering: Event systems use subscriber side filtering mechanism for dispatching event messages to event consumers. The ABDES system enables not only subscriber side filtering, but also publisher side filtering. Therefore publishers can define some criterion for selecting subscribers. Because routing operations are determined by agvents, publisher side filtering is enabled.

ABDES brings a new approach according to the system properties stated above, and covers some open issues of previous distributed event systems. The comparison of ABDES with other distributed event systems is depicted in Table 1.

Systems	Filtering	Filtering	Information	User defined events	Autonomy
Channel based	No	No	No	No	No
Topic based	Only topic	Subscriber	No	No	No
Content based	All fields	Subscriber	No	No	No
Type based	Type, attribute	Subscriber	Yes	Yes	No
ABDES	Type, attribute, behavior	Publisher, Subscriber	Yes	Yes	Yes

Table 1. Comparison of ABDES and other event systems

4 MESSAGE/AGENT FLOWS

Communication between system components 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 transfers are described as follows:

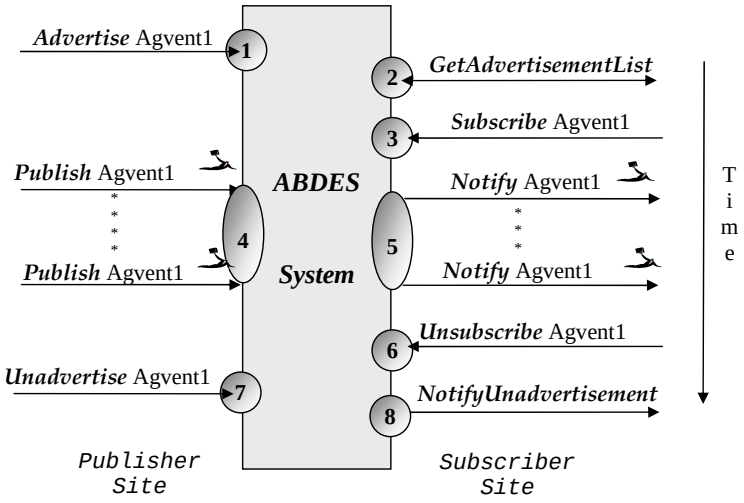

Fig. 3. Message/agent flow of ABDES

1. First, a publisher advertises its agvent type to the system (*Advertise*). This advertisement is dispatched to all agvent servers in the dispatch service via broadcasting.
2. If a subscriber is connected to an agvent server, it can obtain the advertisement list which includes a list of agvent types currently available on the system (*GetAdvertisementList*). All agvent servers hold the same advertisement list.
3. If a subscriber decides to subscribe on an agvent type, it sends a subscription message to the agvent server it is connected to in order to have itself registered (*Subscribe*). Next, this message is routed to necessary agvent servers by the system. The details of the subscription are stored in subscription tables of knowledge bases in each agvent server.
4. Whenever a publisher observes an event, it creates and sends an agvent to the agvent server (*Publish*). When an agvent arrives on an 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.
5. 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 (*Notify*).
6. A subscriber continues to receive published agvents until it issues an unsubscribe request for that particular agvent type (*Unsubscribe*). When an agvent arrives

on a subscriber, it starts to communicate with the subscriber agent according to its pre-defined goals.

7. When a publisher stops publishing a certain type of agvent, it informs the system through an unadvertise message (*Unadvertisemet*).
8. In this case, if no other publisher of that agvent type exists, the system sends out a message to subscribers registered on that agvent type, informing them of the new situation (*NotifyUnadvertisement*).

5 DISPATCHING MECHANISM OF ABDES

The basic publish-subscribe mechanism uses subscribe, unsubscribe and publish primitives for messaging; this is called subscription based routing. A subscription expresses the subscriber's interest in the occurrence of specific events. It is a logical expression that provides the ability to select a subset of events based on their content or type. Subscription messages are used for setting routes for event messages. During the propagation of subscription messages, each event broker behaves as a subscriber with respect to relevant neighbors. Consequently, each of them records the filter associated with the subscription in its own subscription table and forwards it towards the publishers. This process effectively sets up routes for events through the reverse path followed by subscriptions.

ABDES uses an extended version of this mechanism, as used in SIENA [14, 31] by using advertisement messages to optimize the routing of subscriptions. In advertisement-based routing servers use the information provided by event producers (publishers) to route incoming subscriptions. A subscription is only forwarded if it covers the advertisement. Advertisements are used to make agvent types visible to all the participants of the system. Advertisement forwarding, limits the overhead of subscription dispatching by spreading knowledge about agvents throughout the system. When an agvent server receives an advertisement from one of its neighbors, not only it stores the associated agvents' behaviors and attributes into its advertisement table, but it also forwards it to all the remaining neighbor agvent servers, thereby forming a tree to reach all agvent servers. This process effectively sets up routes for subscriptions through the reverse path followed by advertisements.

Dispatching mechanism of ABDES is depicted in Figure 4. First, a publisher advertises its agvent types which it will publish to the system (*Step-1*). The technique of flooding is the simplest approach to implement the advertisement propagation in dispatching service. Here, every agvent server forwards an advertisement that is produced by one of its local clients (publishers) to all of its neighbors, and if an agvent server receives an advertisement from a neighbor, it simply forwards it to all other neighbor agvent servers. Each agvent server exploits a further routing table, called advertisement table, which stores all the received advertisements. If a subscriber decides to subscribe on an agvent type, it sends a subscription message to its connected agvent server, and this message is dispatched to all relevant agvent servers, in reverse direction of the advertisement dispatching tree, by the system

(*Step-2*). Whenever a publisher observes an event, it creates and sends an agvent to the agvent server which it is connected (*Step-3*). 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.

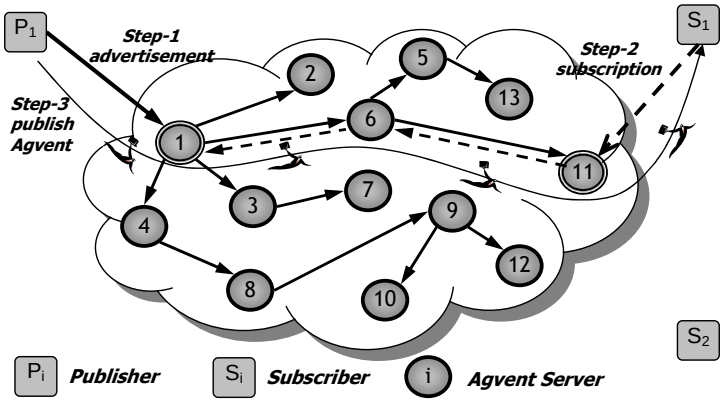

Fig. 4. Dispatching mechanism of agent based distributed event system

6 FAULT TOLERANCE MECHANISM

We assume that agvent servers in the dispatching service of ABDES can fail by crashing and that the failure of a server is eventually detected by all its neighbor servers by using well-know failure detection techniques like heartbeats, or by detecting when there is a necessity. A failed server can cause a gap in the event dissemination tree. To heal the tree, the server which detects the failure re-routes the subscriptions, advertisements and agvents via an alternative route.

Faults and failures are inevitable in a distributed system built over a wide-area network and they should be tolerated for continuing system transactions. Fault-tolerance mechanisms in distributed event systems can cope with different kinds of failures in system middleware and especially integrated with the routing algorithm. This type of fault tolerance results in a scalable and robust system. Researches on fault tolerance mechanism of event based systems are concentrating on two main topics; *self stabilization and reconfiguration*. In [25, 26] authors assume that some links may disappear and others appear elsewhere, because of changes in the underlying. *Reconfiguration* in this case means fixing routing tables entries no longer valid after the topology change. A broker triggers such reconfiguration operation on disappearance of the link with another broker. *Self-stabilization* is an optimistic way of looking at system fault tolerance and scalable coordination, because it provides a built-in safeguard against transient failures that might corrupt the data in

a distributed system. Some event systems [27, 28] use self-stabilization in the broker network by discarding broken and outdated information about neighbors. This methodology is used for synchronizing routing tables because of message loses and is accomplished by the use of leases. Both these models have a side effect as significant increase of the network traffic. Therefore some researches [29] have been done to minimize traffic overhead of the system.

There are only a few researches [30] in the literature regarding node and communication link failures. In these researches, when a broker fails, another broker can take over the role and publish/subscribe model can continue its transactions without any interruption; and when the communication link is disconnected, data transfer is delayed until the link is reconnected. But, if the link is not reconnected then messages over this link cannot be dispatched.

The architecture of event based systems should be tolerant to error and network fallout, especially in dispatching service. Therefore ABDES has a fault tolerance mechanism that can cope with different kinds of failures in the middleware and it is integrated with the routing algorithm resulting in a scalable and robust system. Fault tolerance mechanism of ABDES is thought in three dimensions for faults in advertisement, subscription and agvent dispatching.

6.1 Fault Tolerance in Advertisement Dispatching

In ABDES, advertisements are used to make agvent types visible to all the participants of the system. Advertisement forwarding limits the overhead of subscription dispatching by spreading knowledge about agvents throughout the system. When an agvent server receives an advertisement message from one of its neighbors, it not only stores the associated agvent's behaviors and attributes into its advertisement table, but also forwards this message to all the remaining neighbor agvent servers, thereby forming a tree to reach all agvent servers. This process effectively sets up routes for subscriptions through the reverse path followed by advertisements.

In dispatching of an advertisement, if there is a node or link error and one of the agvent servers (or more) is disconnected, then this agvent server cannot get advertisement messages. This fault causes a disadvantage on routing this advertisement message not to all relevant agvent servers. When this agvent server is online or link is repaired, the agvent server synchronizes its advertisement table by controlling the neighbor agvent servers' knowledge bases. If there are some unsynchronized advertisement messages, these are received from them and forwarded to other neighbor agvent servers (if there is any). This mechanism is also used for adding a new agvent server as a new neighbor to one already in the system. As is to be expected, improving scalability can be seen as more agvent servers are added to the system. Scalability of brokers in an event system is important today. As the Internet continues to grow in popularity and size, dispatching service without good scalability will result in performance drop or live-lock. When a new agvent server joins to the system, agvent servers get the advertisement table of the one of its neighbor and uses it.

6.2 Fault Tolerance in Subscription Dispatching

As usage of advertisements to set up routes for subscriptions, subscription messages are used for setting routes for agvents. In dispatching of subscription, if there is a node or link error in the system (as shown in Figure 5) then the agvent server which detected node/link error should chose an alternative route and forward message on this route. For example if there is a link error between AS9 and AS8 then AS9 cannot forward subscription message over AS8. In this case AS9 can detect the link failure and checks its Alternative Routing Table (ART) for an alternative route to AS1. ART is maintained by each agvent server and composed by every incoming messages (advertisements, subscription and agvents) which also contain their routes to reach their clients. In this failure case the solution can be in two forms:

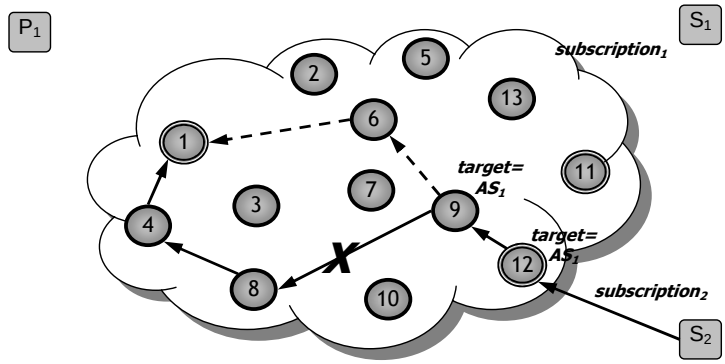

Fig. 5. Fault tolerance in subscription dispatching

- If AS9 contains an alternative route between AS9 and AS1, it forwards this subscription message on this route (see Figure 5). Other agvent servers use these messages for updating their ARTs.
- If there is not any routing information between AS9 and AS1 then AS9 should search an alternative route for it. To achieve this it broadcasts a subscription message whose target is set as AS1 to all neighbor agvent servers. This message is not evaluated as a subscription message except AS1.

In case of adding new agvent server to dispatching service, advertisement table of this new agvent server is copied (synchronized) from one of neighbor agvent servers. But, there is no need to copy the subscription table of the neighbors. Subscription table is composed according to needs of the connected subscribers. In the case of adding an agvent server, two components can join to the system over this agvent server; publishers and subscribers.

- If a subscriber connects to the system over this new agvent server, it sends a subscription message to its connected agvent server and this message is forwarded

according to copied advertisement table. After that the subscription table is updated.

- If a publisher connects to the system over this new agvent server, it sends an advertisement message to the agvent server and this message is broadcasting to all agvent servers in dispatching service. Agvent servers which have connected subscribers interested in this type of agvents reply this advertisement message and forward a subscription message to this newly added agent server.

6.3 Fault Tolerance in Agvent Dispatching

Routes of agvents are decided by subscription messages (reverse path of subscription messages). As shown in Figure 6, the published agvent should be dispatched to S1 and S2. Agvents forward themselves to these destination nodes. After an agvent reaches AS8, a clone of the agvent is created and one is forwarded over AS3, and the other is forwarded over AS9. Because there is a link error between AS8 and AS9, the agvent cannot reach to AS9. Therefore AS8 search an alternative route from its ART. If there is a route, it forwards this agvent over new route.

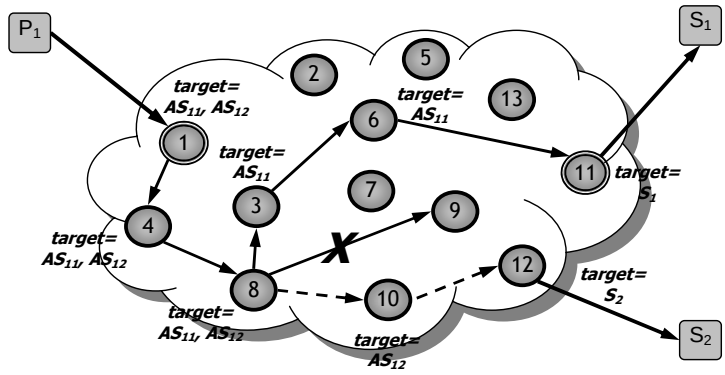

Fig. 6. Dispatching agvents when there is a node/link error

Main problem will still exist if there is not any alternative route. In this case we cannot broadcast the agvent as we do in fault tolerance mechanism of subscription dispatching. An agvent is a collection of code and data. Therefore its size could be from 4–6 kilobytes to 8–10 megabytes (or more) according to its data block. Consequently, we have to establish a new route to send message from broken node to destination node. To achieve this we make a small part of previous messaging processes as follows.

An advertisement is broadcasted to reach the target agvent server, AS12 (sampled in Figure 7). This message is taken into consideration only by the destination server (AS12). Other servers use this message only for updating their ARTs. Destination agvent server replies this advertisement message and forwards the subscription messages about that agvent type. After this subscription message reached

the broken server, an alternative route is set as reverse path of the subscription message. Agvent server re-routes the waiting agvent(s) to destination server(s) over this new route.

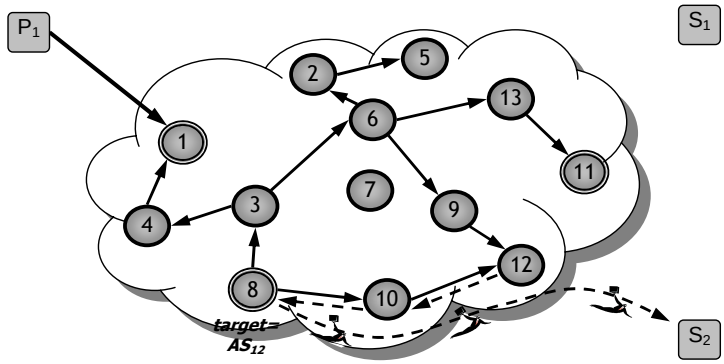

Fig. 7. Fault tolerance in agvent dispatching

7 CONCLUSION

This paper presents a new model for Agent Based Distributed Events Systems, called ABDES, which combines the advantages of publish/subscribe communication and intelligent 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.

ABDES differs from previously developed distributed event systems by its distinctive characteristics like *double side filtering mechanism, autonomous event structure, user defined event types and information hiding.* The system has also the following properties:

Scalable structure: It is possible to increase the number of agvent servers and reach more subscribers.

Expressive subscription model: It is possible to define type, attribute and behavior based filters by subscribers.

Fault tolerant message dispatching: In case of collapsing a link or an agvent server, it is possible to dispatch advertisement messages, subscription messages and agvents from an alternative route.

In order to measure systems performance several tests have been made on ABDES [32]. On comparing these test results we conclude that ABDES has an expressive subscription model and is a scalable system. We think this new model is especially suitable for events which are not generated frequently but contain

large, valuable and secret data for several information-oriented applications, such as e-commerce or information retrieval.

REFERENCES

[1] Microsoft. The component object model (COM) specification web site. Availaible on: http://www.microsoft.com/com/.

[2] JavaSoft. Java RMI. Technical report, Sun Microsystems, Inc web site. Availaible on: http://java.sun.com/products/rmi/.

[3] Gore, P.—Cytron, R.—Schmidt, D.—O'Ryan, C.: Designing and Optimizing a Scalable CORBA Notification Service. ACM SIGPLAN Notices, Vol. 36, 2001, No. 8, pp. 196–204.

[4] Object Management Group: CORBAservices. Common Object Service Specification, Technical Report, Object Management Group, 1998.

[5] Eugster, P.—Felber, P.—Guerraoui, R.—Kermarrec, A.: The Many Faces of Publish/Subscribe. ACM Computing Surveys, Vol. 35, 2003, No. 2, pp. 114–131.

[6] Pietzuch, P.—Bacon, J.: Hermes: A Distributed Event-Based Middleware Architecture. Proceedings of the First International Workshop on Distributed Event-Based Systems (DEBS '02), 2002, pp. 611–618.

[7] Muhl, G.—Fiege, L.—Buchmann, A. P.: Filter Similarities in Content-Based Publish/Subscribe Systems: In Proceedings of the ARCS International Conference on Architecture of Computing Systems, 2002, pp. 244–240.

[8] Strom, R.—Banavar, G.—Chandra, T.: Gryphon: An Information Flow based Approach to Message Brokering. In Proceedings of the ISSRE International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998, pp. 10–21.

[9] Liebig, C.—Cilia, M.—Buchmann, A. P.: Event Composition in Time-dependent Distributed Systems. In Proceedings of the IFCIS CoopIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, 1999, pp. 70–78.

[10] Bornhövd, C.—Cilia, M.—Liebig, C.—Buchmann, A.: An Infrastructure for Meta-Auctions. Second International Workshop on Advance Issues of E-Commerce and Web-based Information Systems (WECWIS '00), San Jose, California, 2000, pp. 21–30.

[11] Segall, B.—Arnold, D.: Elvin Has Left the Building: A Publish/Subscribe Notification Service with Quenching. In Proceedings of the AUUG Australian UNIX and Open Systems User Group Conference, Queensland, Australia, 1997, pp. 243–255.

[12] Fabret, F.—Jacobsen, H.—Llirbat, F.—Pereira, J.—Ross, K.—Shasha, D.: Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Santa Barbara, 2001, pp. 115–126.

[13] Liu, H.—Jacobsen, H. A.: A-TOPSS – A Publish/Subscribe System Supporting Approximate Matching. In Proceedings of the Intl. Conference on Very Large Data Bases (VLDB), 2002, pp. 1107–1110.

[14] Carzaniga, A.: Architectures for an Event Notification Service Scalable to Wide-area Networks. Ph. D. Thesis, Politecnico di Milano, Italy, December 1998.

[15] Lange, D. B.—Oshima, M.: Seven Good Reasons for Mobile Agents. Communications of the ACM, Vol. 42, March 1999, No. 3, pp. 88–89.

[16] Wooldridge, M.—Jennings, N. R.: Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, Vol. 10, 1995, No. 2, pp. 115–152.

[17] Java AWT: Delegation Event Model. (Sun Microsystems.) Availaible on: `http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html`.

[18] Cugola, G.—di Nitto, E.—Fuggetta, A.: The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. Technical Report, CEFRIEL – Politecnico di Milano, Italy, 1998.

[19] SwiftMQ, JMS Enterprise Messaging Platform, Availaible on: `http://www.swiftmq.com/`.

[20] Cao, F.—Singh, J. P.: Efficient Event Routing in Content-based Publish-Subscribe Service Networks. In Proceedings of IEEE INFOCOM, Hong Kong, 2004, pp. 929–940.

[21] Chen, Y.—Schwan, K.—Zhou, D.: Opportunistic Channels: Mobility aware Event Delivery. In ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, 2003, pp. 182–201.

[22] Pietzuch, P. R.: Hermes: A Scalable Event-Based Middleware. Ph. D. Thesis, Computer Laboratory, Queens' College, University of Cambridge, 2004.

[23] Eusgter, P. T.: Type Based Publish/Subscribe. Ph. D. Thesis, Ecole Polytechnique Federale De Lausanne, France, 2001.

[24] Sahingoz, O. K.—Erdogan, N.: Agvent: Agent Based Distributed Event System. In proceedings of 30th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2004), Czech Republic, 2004, pp. 144–153.

[25] Picco, G. P.—Cugola, G.—Murphy, A. L.: Efficient Content-Based Event Dispatching in the Presence of Topological Reconfiguration. 23rd IEEE International Conference on Distributed Computing Systems (ICDCS '03), 2003, USA, pp. 234–243.

[26] Baldoni, R.—Beraldi, R.—Querzoni, L.—Virgillito, A.: A Self-Organizing Crash-Resilient Topology Management System for Content-Based Publish/Subscribe. International Workshop on Distributed Event-Based Systems (DEBS '04), Edinburgh, Scotland, UK, 2004, pp. 3–8.

[27] Xu, Z.—Srimani, P. K.: Self-Stabilizing Publish/Subscribe Protocol for P2P Networks. Distributed Computing – IWDC 2005: 7th International Workshop, Kharagpore, India, 2005, pp. 129–140

[28] Buchmann, A.–Bornhövd, C.—Cilia, M.—Fiege, L.—Gartner, F.—Liebig, C.—Meixner, M.—Mühl, G.: DREAM: Distributed Reliable Event-based Application Management. Springer, 2004, pp. 319–350.

[29] Cugola, G.—Frey, D.—Murphy, A. L.—Picco, G. P.: Minimizing the Reconfiguration Overhead in Content-Based Publish-Subscribe. In Proceedings of the 19th ACM Symposium on Applied Computing (SAC), Cyprus, 2004, pp. 1134–1140.

[30] Oh, S.—Pallickara, S. L.—Ko, S.—Kim, J.—Fox, G.: Publish/Subscribe Systems on Node and Link Error Prone Mobile Environments. ICCS 2005, Lecture Notes in Computer Science, Springer-Verlag, Vol. 3515, 2005, pp. 576–584.

[31] Carzaniga, A.—Rosenblum, D. S.—Wolf, A. L.: Design and Evaluation of A Wide-Area Event Notification Service. ACM Transactions on Computer Systems (TOCS), Vol. 19, 2001, No. 3 pp. 332–383.

[32] Sahingoz, O. K.—Sonmez, A. C.: Mobile Agent Based Publication Alerting System. Lecture Notes in Computer Science, Springer-Verlag, Vol. 3993 ICCS 2006, pp. 903–907.

Ozgur Koray Sahingoz is currently an assistant professor in the Department of Computer Engineering at Turkish Air Force Academy. He graduated from the Computer Engineering Department of Bogazici University in 1993. He received his M. Sc. and Ph. D. degrees from Computer Engineering Department of Istanbul Technical University, in 1998 and 2006, respectively. His research interests lie in the areas of object oriented systems, artificial intelligence, distributed computing, and agent based systems.

A. Coskun Sonmez is currently working as professor in the Department of Computer Engineering at Yildiz Technical University. He graduated from the Department of Electronics and Telecommunications, Electrical and Electronic Faculty of Istanbul Technical University in 1981. He received his M. Sc. degree from Science Institute of Istanbul Technical University in 1983. He received his Ph. D. degree from Cambridge University (UK) in 1992. His major research interests include AI and AI applications, knowledge based systems, expert systems, intelligent automation and fuzzy systems.