Distributed Event Dissemination for Ubiquitous Agents

Sasu Tarkoma
Helsinki Institute for Information Technology, Helsinki, Finland, sasu.tarkoma@hiit.fi

ABSTRACT: Software agent technology is based on event-driven entities that use asynchronous messaging and high-level languages to communicate. Agents may subscribe information from other agents, update their internal models based on received messages and notifications, and publish notifications to listening agents. Currently, most agent architectures do not employ content-based dissemination of messages or notifications, and the agents need to take care of subscription management. In this paper, we examine the benefits of providing a distributed event service for agents that supports mobile components and filtering. We consider using distributed event filtering in enhancing agent functionality and simplifying the agent application development by delegating subscription management to the middleware system.

1 INTRODUCTION

Software agent technology consists of autonomous and asynchronous entities that communicate using high-level languages inspired by the speech-act theory from human-to-human communication [7,16]. Agents may subscribe information from other agents, update their internal models based on received messages and notifications, and publish notifications to listening agents.

Message based communication and message-oriented middleware [23], and event-based systems are well suited to ubiquitous environments [6,17,21,24], because they are asynchronous in nature and message queuing facilitates supporting disconnected operation. The asynchronous and message-based implementation of agent systems bears semblance to the event notification pattern [14]; however, there are differences. Although, agent–based systems are based on asynchronous message passing, say in FIPA ACL (Agent Communication Language), agent messaging is a form of directed communication that targets a select set of recipients. This set of endpoints is determined and selected by the unique identifiers or names of the agents [7].

In this paper, we present and discuss approaches for supporting efficient event-based information dissemination for software agents in ubiquitous computing. Events may be used at two different layers in software agent technology: as a general building block of agent functionality, for example signaling internal state changes, and as a high-level service for supporting content-based routing and complex filtering. Content-based intra-agent routing has been examined in [22]. They present an active message board architecture for the coupling of an agent's components at run-time; however, content-based agent communication in a distributed environment is not addressed. Agent systems would benefit from a distributed agent-based or agent-compatible event service that can support complex filtering, and anonymous one-to-many communication.

An infrastructure that supports a complex filtering language allows the creation of more compact reactive agents. Ideally, agent developers would need to formulate the events of interest using filters, and the code that is executed when situations change or something is requested from the agent. Event-based computing also moves the detection and notification responsibility from the agents to the middleware platform, which also results in smaller, more compact agents and allows the distribution of the filtering process. We also discuss the impact of mobility, and how mobile agents may benefit from a distributed event framework.

This paper is structured as follows: Section 2 presents an overview of distributed event dissemination, and examines the Rendezvous-Notify architecture for mobile computing. Section 3 examines the benefits of distributed events for agents, and outlines implementation issues. Finally, Section 4 presents the conclusions.

2 DISTRIBUTED EVENT DISSEMINATION

In the general model of event dissemination or notification, the subscribers define their interests using filters and strongly typed fields or channel-names [1,5,6,14,18,23,24]. The interest in an event is realized by invoking the interest registration interface on the event service. The event service is a logically centralized component that provides the interfaces for interest expression and notification. The producers or suppliers are responsible for

monitoring particular types of events and notifying the event service.

The two important characteristics for an event service are expressiveness and scalability [5]. Expressiveness deals with how well the interests of the subscribers are captured by the notification service, and scalability deals with federation, resources and issues such as how many users can be supported and how many servers are required.

Event-based systems are becoming a key enabler for distributed applications, because they support asynchronous one-to-many communication and provide a dynamic way to bind components at runtime. However, ubiquitous computing and mobility create new requirements for event-based systems, because of disconnected clients and mobile users [4,5,6,24]. Event service for mobile clients requires support for:

- disconnected operation, and
- terminal mobility, which requires that the subscription tables of event producers are updated in a timely fashion to prevent delivery to old addresses.

Mobility is a relatively new issue in event-based computing, and a few event systems support client mobility, for example JEDI and Siena [4,6]. Mobility support is needed, because clients require that services are provided irrespective of time and place. The Rendezvous-Notify system presented in the next section addresses these requirements and the main emphasis is on the timely completion of handovers and efficient subscription management operations. Many current event systems have a high messaging cost in subscription management, because update messages are propagated to a large portion of the event servers in a distributed system.

2.1 *Rendezvous-Notify*

In this section, we present our Rendezvous-Notify framework, which is an overlay event system for mobile clients based on two server roles: the access server role that is responsible for maintaining subscription information and buffering events, and the resolution server role that is responsible for event channels and routing events to access servers. Resolution servers and thus event channels are located using an explicit-join rendezvous mechanism that uses linear hashing on the event type. The event channel lookup procedure and subscription management has constant or near constant messaging cost, depending on the employed mechanism, and supports enterprise-level scalability. Event domain federation may be used to improve scalability.

The system supports content-based filtering in two phases: the first-phase filtering is done by the event channels, and the second-phase filtering by the access servers. The framework supports relocation of clients by using an efficient handover protocol that moves session data from the old access server to the new access server.

Rendezvous-Notify is currently under development, and a prototype system will be implemented as a generic middleware service that integrates also with agent technology. An agent-based implementation of Rendezvous-Notify would consist of event service agents with two roles: the resolution role responsible for managing event channels, and the access server role. The latter is responsible for buffering notifications to clients and maintaining sessions that group subscriptions together. The FIPA Directory Facilitator (DF) agent would act as a bootstrapping mechanism and the initial lookup point for locating the service agents. In addition, the same explicit-join rendezvous mechanism would be used.

An event service agent may also be executed on the client, if the configuration and system resources support this, in order to provide local event delivery and filtering at the source. Figure 1 presents an overview of a general event framework for mobile computing using agents. The lightweight platform or agent execution environment could be, for example, MicroFIPA-OS [25] or LEAP [3]. W-MTP denotes a wireless Message Transport Protocol (MTP).

The client-server part of the framework, which provides basic tuple-based filtering similar to the Siena filter language [5] and sessions for disconnected clients, has been implemented in Java. We have verified the handover protocol using Promela and Spin [15], and are currently in the implementation phase of the server-side system. The handover protocol is similar to the forward handoff procedure in the Wireless CORBA specification [19]. We plan to use the event framework with the MicroFIPA-OS agent platform intended for small devices [25] in order to support information consuming/producing agents on mobile terminals.

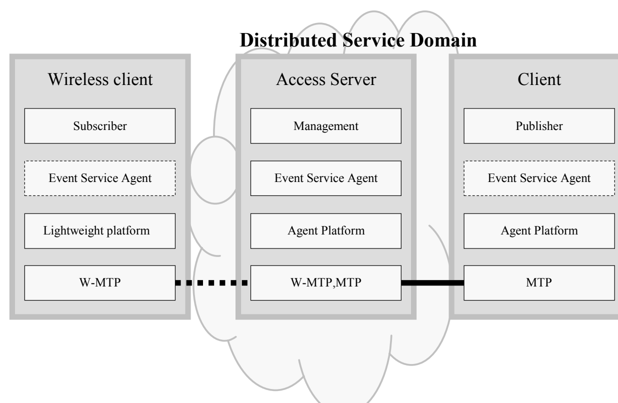

Figure 1. Agent-based Event Service.

2.2 *Event Dissemination for Agents*

Events are published in a named channel, or in an infrastructure of one or more routers that can use the content of the events in making the forwarding decision. Named channels are also called as topics, and they represent an abstraction of numeric network addressing mechanisms. With content-based addressing clients may change their interests without changing the addressing scheme. With channel-based messaging, new channels need to be added to the address space.

The recipients of a message sent by an agent are determined from the receivers-slot in the ACL message, not from the content or type of the message. Current agent communication specifications do not specify support for content-based forwarding of messages. Messages are explicitly sent to a set of recipients, and an intermediate agent or component is needed to manage high-level forwarding of messages. This agent or component may be implemented using standard asynchronous agent messaging and interaction protocols.

Agents may benefit from one-to-many anonymous communication, in which information streams can be filtered and transformed. Moreover, in most current solutions and architectures, agents need to process and manage subscriptions. They need to keep track of the listeners and filter events. By having a distributed event service as a part of the platform or execution environment, agents are spared from the details for subscription management. The composition of reactive agents can be simplified if the platform provides support for filters, and may distribute filters accordingly.

3 AGENTS AND EVENTS

There are several abstraction layers that need to be elaborated: first, we have the basic implementation of a message passing agent that receives messages, and updates its state according to the message type, protocol, contents and possibly the internal model of the world. The implementation of this kind of agent execution environment most likely uses lightweight, internal events to manage different tasks of the agent. With tasks we mean concurrent activities of the agent that may involve messaging. Furthermore, we have a higher layer that subscribes and reasons over information, and produces information for other external entities.

The FIPA architecture [7] aims to improve agent interoperability by providing standards for message transport protocols, message envelopes, agent communication language (ACL), content languages and interaction protocols. In the FIPA architecture the publish and subscribe operations are realized in the fipa-subscribe [9] interaction protocol. This interaction protocol allows the implementation of higher-level observation tasks. Another commonly used protocol that may be used to realize publish/subscribe communication is fipa-request [8]. In this case, the protocol is used by the agent to request another agent to process its subscription or unsubscription.

FIPA does not standardize agent internals. In current agent systems agent functionality is typically implemented using concurrent tasks [20] or behaviours [2], and interaction protocols are implemented as state machines governing the different phases of a protocol specification. Thus the number and nature of tasks and interaction protocols are important metrics in determining the complexity of an agent.

An agent platform implementation may provide some support for conversation management; however, agents still need to keep track of what other agents have subscribed, and when and what to notify them. Agent systems support the Observer-pattern [11], and many consumer agents can register to a supplier agent to receive notifications. In this sense, agent systems can be used to realize event-based frameworks grounded on the Observer-pattern.

The Observer-pattern places the filtering and notification burden on the agents. In many cases, agents on small devices and mobile agents cannot be expected to be able to filter messages and maintain subscription information or large numbers of protocol states. This necessitates the use of a distributed event framework that provides ACL-interfaces or other interoperable interfaces for subscription and filtering.

The Directory Facilitator (DF) specified by the FIPA standardization effort may serve as the initial event service and event type repository, which can be federated across a number of domains. Figure 2 presents an example publish/subscribe example that uses a broker, for example based on the event notifier pattern [14], for subscription management. The example proceeds in the following phases: producer agent locates event channel (1), and publishes events (2). The subscriber agent also locates the channel using DF (3), and subscribes events using a filter (4). Finally, the event channel delivers matching events that are published after the subscription is activated (5).

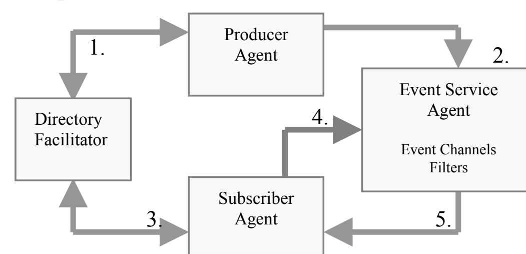

Figure 2. Example scenario with a producer and a subscriber.

We believe that agents would benefit from an intelligent event subsystem, because monitoring and notifying are critical agent tasks. Monitoring and notifying are in many cases distributed tasks, and few agent platforms or models support the creation of distributed tasks for agents. Distributed tasks usually require the use of multiple agents. The benefits of a distributed event service for agents are:

- Reduced complexity of monitoring tasks and thus simpler agents.
- Distribution of detection and notification behaviour and filters. The filtering language determines how expressive tasks can be implemented.
- Anonymous group (one-to-many) communication.
- Resource discovery and synchronization through events.
- Support for mobility and disconnected operation if the event system has this functionality.

The complexity of monitoring tasks is reduced, because the underlying middleware takes care of the complexity of filtering events, subscription management and supporting mobility. If distributed filtering is supported, the filters are distributed within the event system, possibly on a number of hosts. In general, agent communication is directed and the initiator of the communication needs to know the recipient or use a lookup facility in order to determine the destination address. A distributed event service may support anonymous one-to-many group communication through filtering. If the whole content of a notification is filterable, this is called content-based routing of events. In essence, the subscriber does not need to know the set of producers or their addresses, and the events are delivered based on the filters. Anonymous group communication supports resource discovery and runtime binding of components, because the participants in the event-based communication need not know each other beforehand. They need to share an event channel, a topic or a structure in the notification in order to discover each other. Synchronization is achieved using timestamps, causal ordering or total ordering of events.

The distributed event functionality can be provided using different technologies:

- event service may be provided by agents, for example the event service agent (Figure 2),
- a proprietary interface, such as the CORBA Notification Service [18], which is accessed using RPC calls,
- or as new methods in the agent implementation that hide the underlying implementation.

The event service may be implemented, for example, by using CORBA Notification Service or the Java Messaging Service (JMS) [23]. In the case of CORBA, the event agent would provide a FIPA interface for accessing the CORBA event channel, and the functionality necessary for connecting event channels and handling session relocation. JMS and CORBA are general server side architectures for implementing distributed event systems. They do not specify client-side filtering, client-side buffering of events, nor do they support mobility. This motivates the use of an event framework that supports mobility, such as Rendezvous-Notify discussed earlier.

3.1 *Agent complexity: an example*

In order to examine the design of information producing and consuming agents, the structure of these agents may be modeled in terms of tasks, filters, and interaction protocols. A generic information producer agent consists of a set of tasks or behaviours $T_N = \{t_1, t_2, .., t_n\}$. Each task $t \in T_N$ has an associated notification protocol p_t from a set of protocols P, and a mapping M_t between subscribers and filters, given that one task handles one particular type of notification behaviour. The behaviour of each task may be one-shot or cyclic. The information consumer agent is defined similarly, with each task $t \in T_S$ containing a listening protocol behaviour or conversation. Now, the triplet $<T_N, P, T_S>$ defines the information dissemination behaviour of the two-agent system.

An agent using a distributed event service has a simplified structure. An information-producing agent may have a set of information producing tasks, but instead of using multiple protocols for different receivers with a state management overhead, they access only the event service using an interoperable protocol, for example fipa-request and fipa-subscribe, or proprietary means. If proprietary means are used, there needs to be a prior agreement and specification of the API calls, remote invocations or agent interaction protocols that need to be used. The event service is responsible for eventual notification, managing information about subscribers and filters, and ensuring the correct processing order. The information-receiving agent has a set of subscriber tasks that receive notifications from the event service.

3.2 *Ontologies*

One of the benefits of an agent-based event system implementation is the support for logical languages,

such as SL [7], and ontologies [12,13]. One common way to define the concept of an ontology, especially in the context of knowledge sharing, is to call it a specification of a conceptualization [12]. In other words, an ontology is a description of the concepts and relationships that exists for an entity or a community of entities. The main benefit of ontologies is reusability across applications and sharing knowledge. For example: the FIPA device ontology allows agents to reason over device capabilities, such as input and output capabilities and hardware/software components [10].

Ontologies are defined using sets of classes, relations, constraints and other objects. Ontologies are used by agents to represent semantic information. In many cases, ontologies are divided into top-level ontologies, and lower level domain-specific ontologies. Top-level ontologies define general and abstract concepts, such as the notion of time, that are independent of a particular domain.

Ontologies may be used to define events. In a simple case, an event ontology specifies a taxonomy of events, and their structure and properties. This event ontology could be used for automatic filter and interest-profile generation, and event validation. Instead of including the information about the structure of an interesting concept in the application, it is specified, either partially or totally, in the shared ontology. More complex ontologies could include, for example, the pre and post conditions of an event occurrence in a logical language that could be used by agents to subscribe events and update their view of the world upon receiving an event.

Event ontologies and specifications are important in order for information publishers and subscribers to agree on the semantics of an event. In content-based routing an arbitrary filter may match with events from various sources with various semantics. An event ontology or explicit event specification can be used to enforce that all parties have a common understanding of the terms and properties of the events.

3.3 Mobile agents

Mobile agents benefit from using the agent-based event service, because they do not have to process and filter all incoming events. Mobile agents change their location, and a system that has mobile agents as subscribers is not very different from a system that has mobile terminals that roam between access points. Mobile agents and mobile terminals differ in the nature of the mobility; mobile agents usually know the destination server, and the relocation time depends on the size of the agent and the bandwidth and latencies of the communications system. This information can be used to improve the migration of session information. With mobile terminals the new server is not necessarily known and the duration of mobility may be long. On the other hand, if mobile agents are present on mobile terminals, the user mobility characteristics are also present. The Rendezvous-Notify infrastructure has been developed for mobile subscribers, and the same handover mechanism supports the requirements of mobile agents.

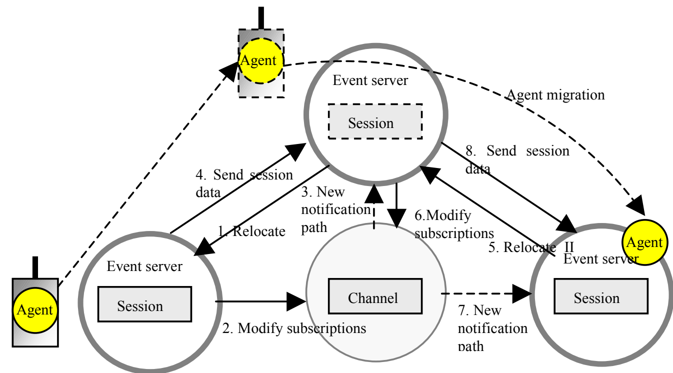

Figure 3. Example of agent and terminal mobility using the Rendezvous-Notify approach.

Figure 3 illustrates both terminal and agent mobility. The event servers in this figure may represent agent domains, or they may all reside within a single domain in order to support scalability. After terminal mobility (1) the system updates event channel subscription information (2,3) and relocates session data to the new server (4). A mobile agent that has subscribed events moves from the terminal to a new server (5). The event channel subscription information is updated (6) and session data transferred to the new location of the mobile agent (7,8). Now, if the mobile agent wishes to return to the mobile terminal, it needs to have a terminal-discovery mechanism available. Note that session data may not necessarily be relocated when a mobile agent moves. The agent could also continue the use of the original event server. Using the event service, it could, for example, publish an event of "discover-terminal" type, and wait for replies that identify the server the terminal is connected to at the time.

4 CONCLUSIONS

We have presented and discussed the similarity and differences of agent technology and distributed event dissemination. Agents are very much based on events and asynchronous messaging that make reactive computation possible; however, agents may also gain benefits from using a distributed event framework. Agent-based systems may benefit from distributed event routing and filtering functionality. The event service can filter, route and store events

on behalf of the agents. The event service functionality may be realized by using a designated event service agent that provides, for example, FIPA-compliant interfaces for subscription management and notification using fipa-subscribe and fipa-request protocols.

Agent implementation is simplified by supporting both client-side and server-side filtering and event buffering. In addition, they do not miss events, and may synchronize through the distributed event service. Mobile agents and agents on mobile terminals benefit from support for disconnection and session relocation between servers.

REFERENCES

[1] Bacon, J., Moody, K., Bates, J., Hayton, R. 2000. Generic Support for Distributed Applications. IEEE Computer, Vol. 33, No. 3.

[2] Bellifemine, F., Poggi, A., Rimassa, G. 2000. Developing Multi-agent Systems with JADE. Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000).

[3] Bergenti, F., Poggi, A. 2001. LEAP: A FIPA Platform for Handheld and Mobile Devices. In Proc. Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL 2001).

[4] Caporuscio, M., Carzaniga, A., Wolf, A. 2003. Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications. Technical report, Department of Computer Science, University of Colorado.

[5] Carzaniga, A.,Wolf, A. 2001. Content-based Networking: A New Communication Infrastructure. In NSF Workshop on an Infrastructure for Mobile and Wireless Systems.

[6] Cugola, G., Di Nitto, E. 2001. Using a publish/subscribe middleware to support mobile computing. In the Proceedings of the Advanced Topic Workshop on Middleware for Mobile Computing, in association with IFIP/ACM Middleware 2001 Conference, Heidelberg, Germany.

[7] Foundation for Intelligent Physical Agents (FIPA). 2003. The FIPA Architecture, Geneva, Switzerland. Available at http://www.fipa.org

[8] Foundation for Intelligent Physical Agents (FIPA). FIPA Request Interaction Protocol Specification. Available at: http://www.fipa.org/specs/fipa00026/

[9] Foundation for Intelligent Physical Agents (FIPA). FIPA Subscribe Interaction Protocol Specification. Available at: http://www.fipa.org/specs/fipa00035/

[10] Foundation for Intelligent Physical Agents (FIPA). FIPA Device Ontology Specification. Available at: http://www.fipa.org/specs/fipa00091/

[11] Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

[12] Gruber, T. 1993. A Translation Approach to Portable Ontologies. Knowledge Acquisition, Volume 5, Number 2, pp. 199-220.

[13] Guarino, N. 1998. Formal Ontology and Information Systems. N. Guarino, editor, Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, Trento, Italy.

[14] Gupta, S., Hartkopf, J., Ramaswamy, S. 1998. Event notifier, a Pattern for Event Notification. Java Report, Volume 3, Number 7.

[15] Holzmann, G. 1997. The Model Checker Spin. IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.

[16] Jennings, N., Sycara, K., Wooldridge, M. 1998. A Roadmap of Agent Research and Development. Journal of Autonomous Agents and Multi-Agent Systems, Volume 1, Number 1.

[17] Raatikainen, K., Christensen, H., Nakajima, T. 2002. Application Requirements for Middleware for Mobile and Pervasive Systems. Mobile Computing and Communications Review, Volume 6, Issue 4.

[18] Object Management Group (OMG). 2002. CORBA Notification Service Specification version 1.0.1.

[19] Object Management Group (OMG). 2003. Wireless Access & Terminal Mobility in CORBA, version 1.0.

[20] Poslad, S., Buckle, P., Hadingham, R. 2000. The FIPA-OS agent platform: Open Source for Open Standards. Proceedings of PAAM 2000, Manchester, UK.

[21] Satyanarayanan, M. 2001. Pervasive computing: Vision and Challenges. IEEE Personal Communications, August 2001.

[22] Skarmeas, N.,Clark, K. 1999. Content-Based Routing as the Basis for Intra-Agent Communication. Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98). LNAI volume 1555, Springer.

[23] Sun Microsystems. 2002. Java Message Service (JMS) specification, version 1.1.

[24] Sutton, P., Arkins, R., Segall, B. 2001. Supporting Disconnectedness – Transparent Information Delivery in Mobile and Invisible Computing. CCGrid 2001 IEEE International Symposium on Cluster Computing and the Grid, Australia.

[25] Tarkoma, S., Laukkanen, M. 2002. Facilitating Agent Messaging on PDAs. Fourth International Workshop on Mobile Agents for Telecommunication Applications (MATA-2002), Barcelona, Spain.