Active Systems (GR/K77068)

October 1995 - December 1998
Jean Bacon, Ken Moody and John Bates*
jmb, km@cl.cam.ac.uk, jb141@eng.cam.ac.uk
University of Cambridge Computer Laboratory
(*since October 1998 at the Department of Engineering)

1 Introduction - technical background

Middleware platforms were designed in the late 1980s to meet the needs of the computer industry, but the recent rapid evolution of technology has caused an explosion in distributed processing requirements. Developers of new applications now routinely expect support for multimedia and mobility of users and computers. Timely response to asynchronous events is crucial for such applications and is not yet adequately supported by current platforms; our extensions for asynchronous operation address this problem.

1.1 Distributed software model

The software model defines the entities that comprise the distributed system, how they interoperate and how their behaviour is specified. An object model is a realistic basis for software system design, including persistent data management, and underpins all recent middleware platforms. The broad applicability of an object model comes from the separation of interface and implementation. A standard for specifying object interfaces allows distributed, independently developed, heterogeneous objects to be composed into a system. For example, the Object Management Group (OMG) and Object Data Management Group (ODMG) provide widely accepted, compatible, standardisation for both transmitted and stored data objects [21, 10]. OMG-CORBA objects are defined using a standard Interface Definition Language (IDL) which allows object names, method names and their arguments to be specified in terms of IDL's base types and constructors. OMG-CORBA supports many programming language bindings; that is, software components written in different programming languages may interoperate.

1.2 Requirements for asynchronous operation

The model on which existing platforms are based is of synchronous method invocation. This is inadequate for many current and emerging applications where asynchronously occurring events should trigger immediate system response. For example, a *cancel* (*credit-card-id*) operation at a banking service may invalidate the record associated with that card at that service and add the card to a hot-list. If the cancellation is because the card is being used fraudulently at this moment, it is important that invalidation should take place at once, and should be notified automatically to all affected services. We are concerned with providing efficient system mechanisms to achieve this style of notification. If synchronous method invocation is all that's available the options are periodic polling and synchronous callback. Frequent polling overloads communications; infrequent polling risks delayed response to individual events. Synchronous callback carries the risk that the caller is held up by any one of the called clients.

Under Active Systems we have developed the Cambridge Event Architecture (CEA). Asynchronous operation is achieved by means of events, event classes, and event occurrences as object instances. For example, an event of type *SEEN* may have data attributes *person, location*, and methods to allow these data attributes to be read and stored.

The term "event" has long been used to describe asynchronous occurrences of interest *intra-application*, for example, within programs to manage user interfaces and sensor systems. In Java Beans, for example, a listener may register interest with an event-raiser as in CEA. However, Java Beans is a proprietary, single-language system and, when a distributed implementation is required, notification is built above synchronous Java RMI. CEA is inter-application and language independent in an open distributed world.

Asynchronous operation supports the following types of application, all of which we have built in prototype form under Active Systems:

- Multimedia support. Control of multimedia may be modelled partly as synchronous method invocation. For example, objects of type *video* have methods *pause, restart, fast-forward, ...* but at application level these invocations may trigger and be triggered by asynchronous events. For example, "when the user clicks on a person on screen as a film is shown, invoke the *pause* method on the video object and show details of the person in a new window"[5, 6, 7].
- Mobility. Mobile users and computers will detach from networked systems and reattach later at some other location. For office applications, users are detected as they move about while wearing an active badge or similar technology [1, 3, 8, 9]. Tourists are detected in a radio networked city. Detection of a mobile user or computer is an event that may require a response from the system [11].
- Group interaction. Suppose a participant in an online conference draws on a shared whiteboard object. The rest of the conference should be able to see the drawing immediately [1, 3, 8, 9].
- Alarms and Exceptions. Security violation (see below), failure of an access control check [2], system overload, the occurrence of an emergency medical condition in a patient, component failure in a network [17], a stock market crash, imminent nuclear power station meltdown are events that require rapid response.
- Management. Network and system management servers must be notified instantly when faults, heavy loading or resource allocation problems occur so that they can take immediate action [17].
- Application composition - active programming. Asynchronous notification may be used as the glue to compose applications. Location technology may be used to inject events into applications or virtual reality visualisations to give augmented reality [4, 9].

1.3 Security

In parallel with Active Systems we have defined an open architecture for secure interworking services (Oasis) [15]. This extends considerably the MSSA capability scheme we proposed in our 1995 application. For scalability, an Oasis server may name its many users in terms of a number of roles; access rights are associated with roles. Oasis needs the CEA's asynchronous notification to support immediate and selective revocation of access rights. Also, Oasis access control is applied both on event registration and on notification, see Section 2.4 below.

2 The Cambridge Event Architecture (CEA)

As outlined above, many emerging applications need asynchronous as well as synchronous operation. Our work has shown how any of the widely used platforms such as OMG CORBA, Java RMI or DCOM can be extended, very simply, to support asynchronous operation. The extension required is that any object, as well as publishing its interface in the standard way using IDL, can also publish, using IDL, the events it will notify to clients if asked.

Such an object has a *register* method in its interface and interested parties may register interest in any event class, specifying parameters or wildcards. Event occurrences are created as objects of specified type. Events may therefore be named and parametrised at a fine grain, using the full range of IDL types, and notified at this granularity. Access control is needed at event registration; a client without appropriate authority may not be allowed to register, and in addition event occurrences to be notified may be subject to restriction by parameter value. When an event occurs, the service matches it against a stored template associated with each registration; subject to access restrictions, each client whose template is matched will be notified of the event that has occurred [14]. Figure 1 shows the scheme. [16] describes how this event extension has been added to Nortel's CORBA implementation. We have since extended Iona's Orbix and ORLs Omni-orb [18]. Other event services have been defined separately [22, 13, 20] rather than as an integrated extension to basic platform mechanisms.

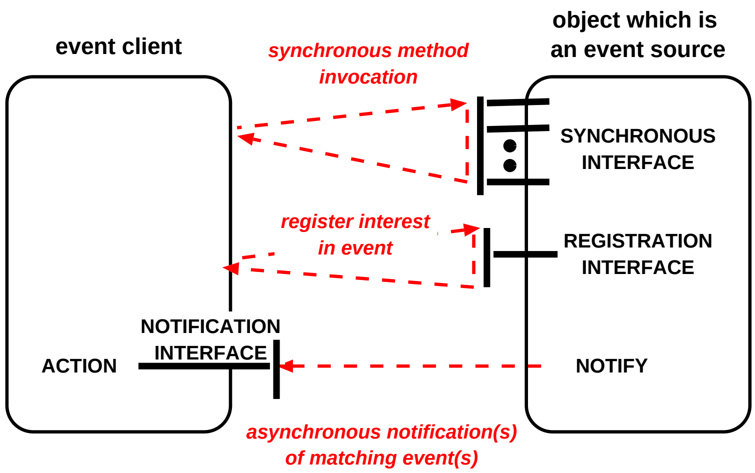

Figure 1: A *publish, register, notify* event architecture

The object publishes its interface, specified in IDL, in the usual way, including the events it will notify. A client may invoke the object synchronously, as usual, and may also register interest in events indicating parameters or wildcards. Events which match the registration template are notified asynchronously to the client, subject to any access restrictions.

2.1 Event mediation

The *publish, register, notify* paradigm can be used for direct, source to client, notification of events. Client-side filtering may be employed, to any extent desired, by using wildcards for parameters on registration. In addition, intermediate services sometimes called **event mediators** can be defined. The architecture is flexible and allows any desired service structure to be built, see Figure 2. An event mediation service might be set up to notify any number of clients and might register interest with any number of event sources.

One use for a mediator is to remove the filtering function from a primitive event source by providing an indirection between it and its potential clients. Any source which cannot afford the overhead of template matching can notify all its detected events to such a mediator. An example is an active badge sensor which detects badge identifiers and notifies all sightings to a higher level, Active Badge Service (in the mediator role). Instead of requiring clients to express interest in an event with parameters (*badge-id, sensor-id*), the Active Badge Service offers an interface which includes, for example *SEEN* (*person, room*).

A mediator can be used to prevent a mobile user from missing events of interest while disconnected from the networked systems for periods of time. The mediator registers interest with the required event sources on behalf of the mobile client and buffers the events notified to it by these sources. It also registers interest in the location of the mobile client and notification of an *attach* event (detecting the mobile user) triggers the delivery of the accumulated events to the user at the new location. Another way of programming this delivery is that the reconnected user polls the mediator, using its synchronous interface, for the accumulated events.

A mediator can provide functionality equivalent to the CORBA event service [21]. It would register interest in all notifiable events with event sources and would provide to its clients both a synchronous, "pull" interface and an asynchronous "push" interface. The recently specified notification service [22] has enhanced the previous event service by defining structured events and filtering at the event service. We are not aware of any implementations as yet.

2.2 Scalability

Our scheme scales well through allowing source-side filtering rather than requiring filtering at a centralised server or at the client. Other schemes multicast events and filter them at the client [20] or offer only mediated notification [22], putting unnecessary load on the communications infrastructure and the clients. Consider the example of tracking users world-wide. It is unacceptable to receive every event about every user, even

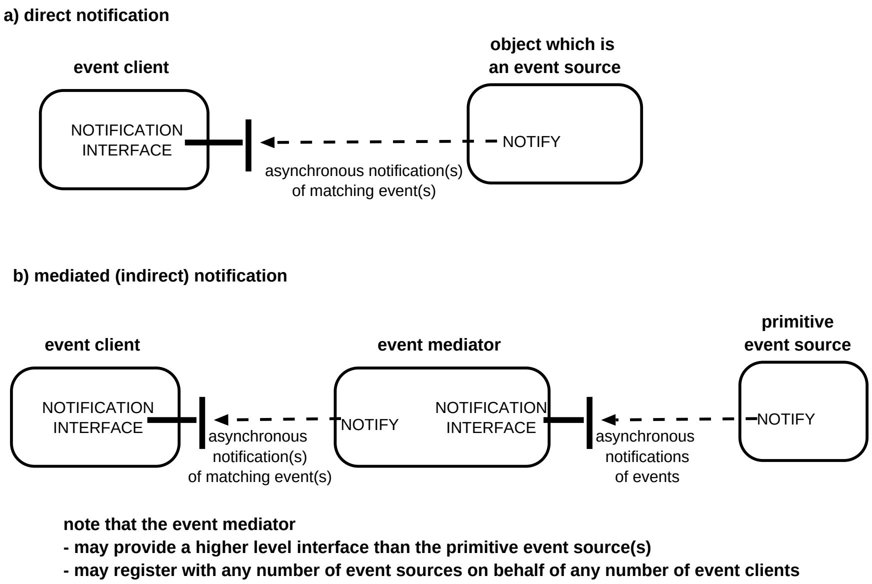

Figure 2: Direct and mediated notification

from a single domain. With source-side filtering events are notified only to clients that have registered interest in the specific, parametrised event. This fine-grained registration is the only way to keep network costs and client filtering at an acceptable level. Note that the search complexity for template matching is limited by the number of parameters, and therefore scales well with the number of clients as well as minimising the use of communications bandwidth.[16] includes performance measurements.

2.3 Composite events

An extension of the simple, basic event paradigm is to combine primitive events into composite events. Complex scenarios can then be defined and specified and composite event services can be built which will, on their clients' behalf, register interest with appropriate event sources and notify clients when a composite event is detected. A composite event server is an example of an event mediator which can carry out filtering across events of different types from different sources.

There are many possible application scenarios; for example, a medic may ask to be notified if the temperature of a patient rises by more than one degree within 24 hours of a new drug being administered via a drip. An active badge server may be asked to notify when more than two members of the Opera research group are in any of the Lab's meeting rooms on any Friday after 11am. Fault detection in a communications network may be carried out across multiple components of different types [17].

We have defined composition operators, with a small language for specifying composite events, and use finite stream semantics [12] to model the arrival of events from the various event sources which are components of distributed systems, see Figure 3. The operators are:

without	A-B	yields the stream matching A until B occurs,
sequence	A:B	yields the stream matching B after A occurs,
or	AorB	yields the stream matching A or B,
and	AandB	yields the stream matching A and B,
First(A)		yields the first occurrence of A.

We have built an active home demonstration to illustrate these concepts, see the Opera group web pages.

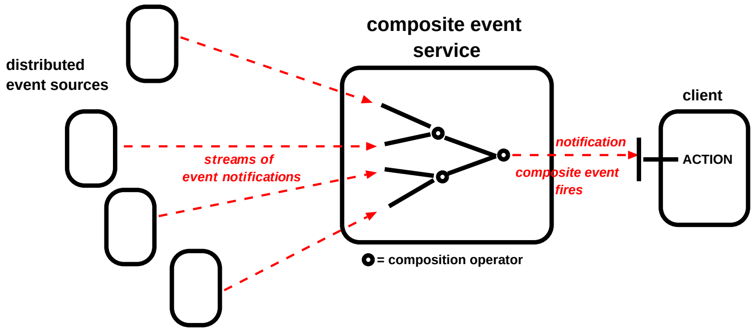

Figure 3: Composite event detection

2.4 Interdependence of Oasis and events

Oasis is built above our asynchronous event infrastructure. During authentication of a principal into some role of a service, the service will interact with any other services that are specified in the role entry conditions. First, each certificate presented as a credential must be checked by the service which issued it using an encryption check. If all is well, event channels are then set up between the services via the *publish, register, notify* mechanisms of CEA. As we have seen, notification of a security exception achieves immediate invalidation of any dependent certificates. See [15] for a full description of Oasis.

2.5 Distributed implementation issues

A fundamental characteristic of distributed systems is that nodes and links may fail at any time. Components of a distributed software system may fail, or become unreachable but continue to work, and we cannot be certain which is the case.

The absence of an event notification means either that the event has not occurred or that there has been a failure in the relevant source or its connection. In this case the event might have occurred but the notification cannot be sent or received. For this reason we use a **heartbeat protocol** between event sources and clients, both of which may be mediators. The period at which a heartbeat is sent is tuneable pairwise between source and sink. In the absence of an expected heartbeat each may specify its own policy. For example, an Oasis server with high security requirements might suspend the affected access control certificates until heartbeats resume. Heartbeat rates are negotiated on a source to sink basis. Messages that assert continuing life need only be sent when there is no interaction between two parties.

System failure means that events in main memory awaiting transmission may be lost. Heartbeats alert the client to this possibility. If reliable transmission is required by an application the events must be stored persistently so that they can be recovered on restart. Our automatic stub generation, based on ODL, facilitates event storage as well as transmission [23].

3 Resources

Staff were employed as specified in the proposal. The grant was extended by three months to allow for a gap in recruitment between RA1s and maternity leave of RA2. Two PhD students not employed on the grant contributed to the research on location-aware computing and event storage and retrieval.

The experiments on multimedia control, location-aware computing and groupware were carried out at the computer laboratory using the DEC Alpha station purchased on this grant and other earlier systems. Infrastructure support was needed for maintenance and file system support for the project.

We have been successful in publishing the work at International workshops and conferences, see the bibliography. The travel grant was spent partly for this purpose, as specified. We visited Nortel regularly and gave presentations there and at other UK and US universities and companies, see the RGR1 form. We supplied six-monthly reports on the research to Nortel as well as reports on specific topics [17, 18, 19].

References

[1] J Bacon, J Bates and D Halls, “Location oriented multimedia” *IEEE Personal Communications* 4(5), pp 48-57, Oct 97

[2] Bacon J, Hayton R, Moody K, “Middleware for Digital Libraries” *D-lib magazine, the magazine of digital library research*, October 1998, http://www.dlib.org/dlib/october98/10contents.html

[3] J. Bates, J. Bacon and D. Halls, "Middleware Support for Mobile Multimedia Applications”. *ICL Systems Journal*, November 1997.

[4] Bates J, Bacon J, Moody K, Spiteri M, “Using events for the scalable federation of heterogeneous components”, *Proceedings of the ACM SIGOPS EW 98*, Sintra, Portugal 1998

[5] Bates J and Bacon J, “Supporting Interactive Presentation for Distributed Multimedia Applications”, Multimedia Tools and Applications (Kluwer), 1(1) Feb 95

[6] Bates J and Bacon J, Chapter: “Design and Applications of Multimedia Systems” in edited book Multimedia Systems and Applications Vol II Multimedia Tools and Applications, Kluwer, 1995

[7] J Bates, “Middleware Platforms to Support Multimedia Applications”. In Book Handbook of Internet and Multimedia Applications, Edited by B. Furht *CRC Press,* 1998.

[8] Bates J, Halls D, and Bacon J, “A Framework to Support Mobile Users of Multimedia Applications”, *ACM Mobile Networks and Nomadic Applications (NOMAD)*,1(4), pp. 409-419, 1996

[9] J Bates, M Spiteri, J Bacon and D Halls, “Integrating Real-World and Computer-Supported Collaboration in the Presence of Mobility”, *Proceedings of IEEE Workshops on Emerging Technologies in Collaborative Environments (WET'ICE) 1998, Workshop on Collaboration in the Presence of Mobility*, Stanford CA, June 1998

[10] R G Cattell, *The Object Database Standard ODMG 2.0*, Morgan-Kaufmann 1997

[11] N Davies, K Mitchell, K Cheverst and A Friday, “Caches in the air: disseninating tourist infornation in the Guide system” *proc. workshop on mobile computing systems and applications*, New orleans 1999

[12] W Ferreira, “A stream-based semantics for composite events”, Internal report of the Opera group, University of Cambridge Computer laboratory, 1997

[13] R E Gruber, B Krishnamurthy and E Panagos, “READY: A notification service for Atlas”, *Tech report AT&T Labs - Research*, 180 Park Avenue, Florham Park, NJ 07932, Sept 97

[14] Hayton R, Bacon J, Bates J and Moody K, Using Events to Build Large Scale Distributed Applications., In *Proceedings of the ACM SIGOPS EW 96*, Connemara Ireland, 1996

[15] R Hayton, J Bacon and K Moody, “OASIS: Access Control in an Open, Distributed Environment” *Proc IEEE Symposium on Security and Privacy*, pp 3-14, Oakland CA, May 1998

[16] C Ma and J Bacon, “COBEA: A CORBA-Based Event Architecture”, *Proc USENIX COOTS'98*, pp117-131, Santa Fe, New Mexico, USA, Apr 1998

[17] C Ma, “An alarm correlator based on the Cambridge event technology” Internal report, 1997

[18] C Ma, “A comparison of the Nortel ORB, Orbix and OmniORB”, Internal report, 1998

[19] C Ma, “A comparison of the CORBA notification service and the Cambridge event architecture”, Internal report, 1998

[20] K O'Connell and V Cahill, “System support for scalable distributed virtual worlds” *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM, pp 141–142, July 96

[21] OMG, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, July 1995

[22] OMG: NEC et al., *Notification Service*, OMG TC Document telecom/98-06-15, June 1998,

[23] M Spiteri and J Bates, “An Architecture to support Storage and Retrieval of Events”, *Proceedings of MIDDLEWARE 1998, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp 443 - 458, Lake District, UK Sept 98