Global Computation using Events (GR/M22413)

October 1998 - December 2001
Jean Bacon and Ken Moody
jmb, km@cl.cam.ac.uk, http://www.cl.cam.ac.uk/Research/SRG/opera
University of Cambridge Computer Laboratory

1 Overview

Throughout the period of the grant the investigators have continued to lead research in the design and deployment of large-scale, widely distributed applications. The work includes the closely related areas of event-driven applications (originating from the Cambridge Event Architecture, CEA), (Oasis) role-based access control (RBAC), and runtime application control based on active policy stores. GR/M22413 has enabled significant investigation and deployment of the event-driven paradigm. Two other small grants have each supported a graduate student in the related areas of RBAC and policy management. The grant-supported, and other, PhD students work together as a group.

Alexis Hombrecher was the attached student on GR/M22413 and is writing a thesis on "Federation for reconciling event taxonomies across administrative domains". His work contributed to objectives 1, 2 and 3, as descibed in sections 2, 3 and 4 below and section 5 describes his and others' work on unifying event transmission, storage and querying. Section 6 describes the integration of CEA and Oasis (objective 4). Section 7 describes the implementation of CEA on a variety of middleware platforms (objective 5). Section 8 covers event-driven application control (objective 6). Section 9 outlines other research in the area of the grant and the appendix lists our currently submitted and further publications.

2 Event-based federation, introduction and background

The first objectives of this research were to establish a new event paradigm that allows connecting new, as well as existing, event architectures in order to build a global computing environment. Prior to this work, event systems had been considered as stand-alone systems without an interface to applications outside their domains. It was the aim of this project to establish a paradigm and design and build an architecture that would allow not only to link individual applications via an event architecture, but also to connect event systems. We have therefore created and implemented the event federation paradigm. Here we will describe this paradigm and our implementation of it.

Event federation combines the usually disparate areas of event systems and system federation. One use of an event system is to connect heterogeneous applications by sending notifications of an occurrence of some event to interested parties. This implies that event middleware establishes a connection between two or more applications. The focus is not on connecting different types of event systems, but rather heterogeneous applications.

Database federations on the other hand try to provide a uniform means of linking different types of databases. It is important that new databases can join existing federations, relatively easily and without much programming effort. Loosely and tightly coupled federations must be distinguished; in a loosely coupled system a database retains a high degree of autonomy, and vice versa.

Federation can be broken down into two separate disciplines. The first entails resolving system and application heterogeneity. This means getting a machine of type A running database X on a particular OS to communicate with machine of type B running database Y on yet another OS. The second discipline focuses on resolving semantic heterogeneity between two different systems. This involves trying to reconcile and understand what different systems mean when they describe the data they maintain.

Classical database federation is described in terms of a five-level schema architecture [11]. This represents a tightly coupled architecture and can be restrictive because it requires all systems wishing to participate in the federation to comply with a global schema. More loosely coupled approaches [10] partly deal with this issue. With those approaches the focus is on resolving semantic heterogeneity, rather than focusing on restructuring data.

The concept of heterogeneity, specifically structural semantic heterogeneity, thus becomes very important. It deals with the difference in meaning of descriptions of entities within a system. Resolving this type of heterogeneity is essential for allowing different systems to communicate. The main forms of structural semantic heterogeneity are:

- domain conflicts - the same object or entity is described differently in different domains
- naming conflicts - the same attribute has different labels
- type (or meta-data) conflicts - different types are used to describe related entities
- structural conflicts - different data organisation or structure to represent the same concepts

3 The Event Federation Paradigm

CEA events were originally typed using IDL for transport between heterogeneous components. The type system needed to be extended for storage (and query) and to describe the event types in a semantically rich manner. We first used the Object Definition Language [1] (ODL) to describe events. We were able to develop class hierarchies using ODL. Deriving specific instances from a common supertype allowed us to resolve some of the difficulties encountered by heterogeneity. The approach proved too difficult when trying to connect very different event systems with their own unique type system. We resolved this problem by using XML Schema [4, 5] to define event types. We found that any type system could be expressed using this schema definition language.

Furthermore, we decided to make minimal assumptions on how our world is constructed. In our view, the world consists of a set of domains. A *domain* is a logical scope of a collection of event sources where a single administrative power is exercised. The event sources in a domain may be related, but this is not a requirement. A domain represents a unit of autonomy. The domain administrator has full control over its event sources and types defined by each source.

Domains reduce the degree of complexity of a distributed system by partitioning it into smaller, more manageable entities. Having established the notion of domains, we further developed this idea to model interworking domains, which we call a *federation*. In our paradigm, we use the term *federation* to refer to multiple, independent but cooperative domains, joined by a set of agreements, or *contracts*. Domains must strictly conform to the agreement a federation defines in order to enable it to participate. This might imply that an authority of some kind exists, but this is not necessarily the case.

3.1 Domains, federation and contracts

A federation is created when at least two domains agree on at least one contract. Once a contract has been defined between two or more domains, other domains can join the federation if they fulfill certain requirements. This will be determined by a federation administrator, typically an existing domain. The administrator is responsible for handing out federation identifiers for each domain wishing to partake in the federation. The identifiers are unique to the federation.

There are two possibilities for a domain that wishes to join a federation. Either it will join an already existing contract, or it will create a new contract with an existing member of the federation. One may also envision a system whereby two completely new domains with a new contract want to join the federation. Such a scenario may be desirable, especially when the new contract type is of interest to other members of the federation. This is at the discretion of the administrative domain.

It is important to note that each domain participating in a federation must have identical gateway components for communication with other gateways. Each domain must have at least one gateway to participate in a federation, but may have multiple gateways. A gateway may only participate in one federation, but a domain may participate in many federations with multiple gateways.

In this architecture a domain is a logical grouping of event services, which is autonomous. Domains agree to share information with other domains by publishing event types they are willing to share. Two or more domains wishing to communicate establish a contract. The notion of a contract is derived from real-life examples where entities interact with each other. Conceptually, a contract is a binding agreement between two or more parties. We use the term contract to mean an agreement between two or more cooperating domains. A more formal definition of a contract is given below:

> Let T_A be an event type defined by a source S_A of domain A, and T_B be an event type defined by a source S_B of domain B. We can define sets E_A and E_B as the sets of all instances of T_A and T_B respectively. A contract between A and B with respect to T_A and T_B is defined as an event type $\mathcal{T}$, whose set of all instances is denoted as $\mathcal{E}$, and which satisfies the condition that there exists a function $f : E_A \to \mathcal{E}$ for A and $g : E_B \to \mathcal{E}$ for B.

The function that maps a local event to an event of the type defined by a contract is called a *schema translation function*. A schema translation function that translates an event from type T to type S is a translation function from T to S. Schema translation takes place at the intersection of two domains, the gateways. An event, which is going to be sent across domains, will be translated within its domain of origin. This means the foreign domain always gets events conforming to a contract it has agreed to. One advantage of the elegance and simplicity of this model is that applications are easier to build. Applications only need to handle the types that they are familiar with, so that dynamic handling of foreign types is unnecessary. Another advantage is that this model effectively prevents the details of an event local to a domain being exposed to the foreign domain.

3.2 Gateways

A gateway acts as an event producer or an event consumer, depending on whether it is handling incoming or outgoing events. Events coming from another domain are translated into the local event type according to the translation function of the contract and are then passed to the event consumers within the local domain. Local events are translated into the contract type and sent to the domain(s) participating in the contract. This means that the gateway manages a possibly complex set of contracts and appropriate translation functions.

A gateway manages all existing contracts with other domains. Furthermore, it publishes the contract types it currently has available for cross-domain event discovery; it identifies a particular event type and translate it using the correct translation function; it propagates events to the correct entity, either within its domain or to other gateways; it negotiates new contracts with other domains; it translates event registrations to and from the contract registration; it registers interest with services within a domain; it communicates with foreign domains; and it manages interaction with foreign domains based on specific security policies.

4 The Architecture

XML [2] is the core technology used to connect the different components in our architecture. It serves as the foundation for supporting interoperation among heterogeneous domains. This is an important consideration for an open architecture, designed to support global computation. The rationale behind our choice of XML is thus clear:

- XML is open and extensible. This enables features to be added to our architecture without breaking the interoperability. This means local customisations can be made by applications if so desired.
- XML uses a text representation which imposes fewer restrictions on application implementations. Applications can be programmed in any programming language, and any transportation protocol capable of sending text can be used.

We make extensive use of XML in our architecture. Event Markup Language is the language in which we represent events. Its schema is defined using XML Schema [4, 5], and applications may define domain-specific event types by extending our language. Registration of interest is expressed in Registration Markup Language. This language was designed with expressiveness in mind. It supersedes the conventional pattern-style of registration in several ways.

We have investigated the use of SOAP [6, 7] for representing event instances. Although this is a viable alternative and may be integrated in future implementations of our architecture, we feel that our current language has all the features we require. The body of any SOAP message is in fact very similar to our representation and we therefore have not felt the need to incorporate SOAP. Similarly, we have experimented with WSDL [8] for representing the event services available within a domain. Again, we felt that simply using XML Schema was as expressive as WSDL and hence decided to continue with our solution. One argument in favour of WSDL is that it provides a well defined interface for querying about available services.

We have also considered using XPath [3] for registration of interest, rather than our registration language. We are still investigating this, especially as a new specification for XPath is about to be released. Many problems we have encountered with XPath are likely to be addressed in this specification.

Our use of XML to represent events leads to the natural choice of XSL Transformation (XSLT) [9] for the implementation of schema translation functions. XSLT is designed for the generic transformation of XML. We have not experienced any situation where its power was not able to handle our needs in schema translations.

5 Integration of event transmission and storage

Event storage can occur at two levels. Firstly, it can be embedded within the event source or client. Secondly, dedicated storage nodes can be used, providing an explicit service. The first is useful for services like buffering at the source or client, while the second is useful for analysis of event histories. In the case of event history analysis, we view event stores as consumers of events. A storage component registers interest with a source when it plans to store event occurrences. It then receives events and stores them in the database.

Our architecture allows all components, including mediators, event stores and event gateways to use the standard registration and notification services. Thus, selected, filtered events can be logged and subsequently queried. We used ODL as an event type system, an ODMG database as an event store and OQL as the query language. Alexis Hombrecher and Oliver Seidel built the ODMG event store and a student project implemented an ODMG browser for it, see appendix ref [16]. Theses in this area were Seidel's "Metadata support for connecting application components asynchronously" and Mark Spiteri's "An architecture for the notification, storage and retrieval of events". A database is needed for storing large volumes of event occurrences and querying (mining) for patterns of events, for example for fraud detection.

Our recent work explores the use of XML for event typing, both inter and intra domain. XML is becoming widely used but is not suitable for full-scale database implementation. However, if one is only interested in storing event meta-data and small amounts of event data for the purpose of delivery guarantees or mobile computing then a simple XML store such as dbXML could be appropriate.

6 Integration of CEA and Oasis RBAC

Under a related grant Walt Yao has implemented Oasis RBAC using web service technology and XML messaging. The authorisation policy associated with any service invocation, including registering interest in events, can require a role membership certificate (RMC) as a credential. Oasis requires a monitoring framework, a so-called "active security platform". Event channels are set up between services so that, should any required security condition become false, an event is sent, causing any dependent role to be deactivated and any dependent ongoing service invocation terminated, see appendix ref [1].

7 Implementation of CEA on standard middleware

Dr Ma has managed this part of the work. By setting up final year undergraduate projects she caused CEA to be integrated with a range of CORBA and Java-based middleware, see appendix refs [16 - 23]. The students benefitted greatly from working with a research group and one, Peter Pietzuch, is now a graduate student with us. The work is summarised in a paper submitted to Software Practice and Experience and a more comprehensive technical report is being written. Software is available from our research group web page http://www.cl.cam.ac.uk/Research/SRG/opera/ which extends CORBA with event services and uses ODL (object definition language) as a basis for event transmission and storage.

8 Event-driven application control

Our work in this emerging area will continue for some time. Andras Belokosztolszki, Dr Moody's graduate student, extended Postgres with database triggers and its use as an **active policy store** seems highly promising, see appendix ref [11]. We currently have an implementation of access control policy and an active security platform, as descibed above.

Glasgow Caledonian University, with Strathclyde, have an EPSRC grant (Jan 2002 - Dec 2003) to implement evolving tertiary courseware. This application needs both RBAC and an event-driven workflow, and they propose to use Oasis and our policy store. The grant contains travel and subsistence support for two-way visits to facilitate this transfer. The first visit was Nov 21st and 22nd 2001.

Alan Abrahams started his PhD in 1999 in the area of e-commerce. He is investigating the event-driven paradigm for driving these applications, using business contracts as the basis of their specification. His policy store, in which the essentials of the contracts are stored, currently uses Microsoft Access and triggers are implemented as a wrapper, which degrades performance. We have applied for funding for further work in this area.

9 Other research in the area

Peter Pietzuch's thesis work (Hermes) extends CEA for high volume, wide-area event transmission. The approach is complementary to our previous work in that it supports loosely coupled, asynchronous services, that may detach or be down for periods of time. The use of XML as an event type system allows automatic generation of event types from programming language types. In this respect, XML is superior to the naming schemes employed for subject and topic based publish/subscribe systems (concatenated strings) such as the TIBCO rendezvous.

Hermes exploits a peer-to-peer overlay network for load balancing (scalability), see appendix ref [9]. Profs. Jon Crowcroft, George Coulouris and Dr Bacon propose to contrast this approach with one based on IP multicast, thus integrating event and communications services. A grant application is about to be submitted.

Agilent have awarded an international fellowship to support a PhD student from October 2000. He is working on event systems for telecommunications management. Dr Joe Sventek, of Agilent Edinburgh, is the UK contact. Dr Sventek has long experience in Middleware, starting from the ANSA project in the 1980's and he has recently worked on publish/subscribe systems [12].

CEA has been used in the Home Area Network project at the Computer Laboratory, see http://www.cl.cam.ac.uk/Research/SRG/netos/han/second.html. Event subscription and notification are part of an embedded operating system, UbiqtOS, and event publication is supported through a directory service, see appendix refs [5,6,7].

10 Resources

The staff budget was used as planned for Chaoying Ma as RA and Alexis Hombrecher as attached PhD student. Dr Ma was not available until January 1999 and left in October 2001. Mr Hombrecher continued as RA for the final three months. The travel budget was used as planned for conference attendance and visiting groups active in the area, such as Professor Buchmann's group at Darmstadt. The desktops we had long used failed during the period of the grant and were replaced with PCs.

11 Summary

The objectives were fully achieved and additional research was carried out, since several PhD students were attracted to work in the area. Our starting point was to add asynchronous operation to synchronous middleware, such as CORBA and the Java family, and to investigate the programming of event-driven applications. Our selection of XML, initially to support inter-domain message transport and contract specification, fitted well with the widespread adoption by the commercial world of XML for message transport, for example IBMs MQSeries messaging. We therefore extended our work to use XML for event services in loosely coupled asynchronous applications and thus facilitated their programming.

The PI has initiated, with the Darmstadt group, a workshop associated with ICDCS which is exclusively on distributed event-based systems; the number of submissions is high. The event paradigm is becoming widely accepted internationally.

The grant has stimulated a great deal of interest, much ongoing activity in the area and proposals for further research.

References

[1] R. G. G. Cattell, D. Barry, M. Berler, J. Eastman, D.Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez. *The Object Database Standard: ODMG 3.0.* Morgan Kaufmann Publishers, San Diego (CA), USA, 1999.

[2] World Wide Web Consortium. Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[3] World Wide Web Consortium. XML Path Language (XPath), 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[4] World Wide Web Consortium. XML Schema Part 1: Structures, W3C Working Draft, 25 February 2000. http://www.w3.org/TR/2000/WD-xmlschema-1-20000225.

[5] World Wide Web Consortium. XML Schema Part 2: Datatypes, W3C Working Draft, 25 February 2000. http://www.w3.org/TR/2000/WD-xmlschema-2-20000225.

[6] World Wide Web Consortium. SOAP Version 1.2 Part 1: Messaging Framework, 2 October 2001. http://www.w3.org/TR/2001/WD-soap12-part1-20011002.

[7] World Wide Web Consortium. SOAP Version 1.2 Part 2: Adjuncts, 2 October 2001. http://www.w3.org/TR/2001/WD-soap12-part2-20011002.

[8] World Wide Web Consortium. Web Service Description Language(WSDL), 15March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[9] World Wide Web Consortium. XSL Transformations (XSLT) Version 1.1, 24 August 2001. http://www.w3.org/TR/2001/WD-xslt11-20010824.

[10] Joachim Hammer and Dennis McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *Journal for Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.

[11] A. P. Sheth. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. In *International Conference On Very Large Data Bases (VLDB '91)*, pages 489–490, Hove, East Sussex, UK, September 1991. Morgan Kaufmann Publishers, Inc.

[12] P, Eugster, R. Gerraoui and J. Sventek. Distributed asynchronous collections: abstractions for publish/subscribe interaction ECOOP 2000, LNCS pp252-276, Springer 2000.