A Composite-Event-Based Message-Oriented Middleware

Pingpeng Yuan and Hai Jin

Huazhong University of Science and Technology, Wuhan 430074, China
yuanpingpeng@163.net, hjin@hust.edu.cn

Abstract. There is an increasing interest in tying together software systems to interoperate and cooperate over the Internet. One common glue technology for distributed, loosely coupled, heterogeneous software systems is *Message Oriented Middleware* (MOM). In this paper we present a framework for composite-event-based MOM (cMOM) and its implementation. The cMOM can also be considered as a grid event service, which is a basic component needed by many grid services. cMOM allows distributed applications registered their interests in composite-events. When composite-events occur, cMOM notifies responding applications and those applications performed action. Our framework comprises an execution model – the event-service interacting model. The paper discusses some aspects of the event-service interacting model. The novel aspect of our cMOM lies in supporting composite event and its detection and enabling the integration of services within and across organizations without requiring changes in other applications.

1 Introduction

Advances in networking and the pervasive deployment of the Internet have created new opportunities of using distributed computers as a single, unified computing resource, leading to grid computing [1]. Early research on grid computing mainly concentrates on computational grid and data grid. With the development of e-Business, the requirement of using applications available on Internet as a single, unified application, namely business grid, is also impending.

The key problem in business grid is how to glue applications distributed over Internet. The basic observation about applications is that widely disseminated events (or messages) are becoming ubiquitously available through the Internet, mobile devices. These public events, as well as internal events can form the *glue* to link applications within and across organizations. As this technology is based on messages, thus, *message oriented middleware* (MOM) naturally becomes one of the most prospective gluing technologies. Using MOM environment to glue together a large number of stand-alone applications, each application may evolve independently from others in this environment. The MOM environment will allow new applications to *tap into* information generated by existing applications without disturbing them.

The remainder of this paper is organized as follows: First, we review some related works and evaluate some common MOMs found in the literature. Then, we present a new approach to MOM: composite-event-based MOM and its implementation. Finally, we end this paper with conclusion.

M. Li et al. (Eds.): GCC 2003, LNCS 3032, pp. 700–707, 2004.

2 Related Work

Message oriented middleware (MOM) is a commonly used to glue distributed applications. Many attempts have been made to gain more and more expressive power and flexibility in MOM. In the following, we introduce some research briefly.

The CORBA Event Service [2] and Notification Service [3] components are defined in the *CORBA Object Services* (COS) layer. The most important feature introduced by the two services is to enable each client to subscribe to the precise set of events it is interested in receiving.

Examples of MOM products include MQSeries [4], MSMQ, Active Web, Message Q, TIB/Rendez-vous, Open MOM, Pipes, etc. More detail comparison of those products can be found in [5]. Here we only introduce the representative product - MQSeries briefly. MQSeries allows applications to communicate using messages and queues. However, MQSeries treats events as uninterpreted data and cannot perform content-based publish/subscribe. Moreover, it can not deal with composite events or complex messages.

In contrast to MQSeries and CORBA Notification Service, the Gryphon project [6] is an advanced technology of MOM and extending its range of application. Gryphon introduced the following extensions: content-based publish/subscribe, stateless event transformations, and event stream interpretations. Gryphon's approach to event distribution middleware is based on the concept of *information flow graphs* [7, 8].

Although those research mentioned above are carried on MOM, current MOM reveals that these systems or specifications have two main limitations:

1) Provide no methods to express composite events. In many legacy applications, a set of predefined events has been provided. Although it is possible to glue applications together using primitive-event-based MOM according to those predefined events, supporting only a small number of predefined events severely limits the ability of MOM to link complex applications where temporal and external events need to be combined with internal events.

2) Lose trace of execution of the application and cannot rollback tasks at any level. The execution models those MOMs based on are too simple; therefore, it cannot record the execution of applications and maintain the states of all executing tasks. When exceptions occur or users want to abort tasks, those MOMs cannot correctly rollback all or specified tasks.

3 Composite-Event-Based Message-Oriented Middleware

The above observations lead us to explore *Composite-event-based Message-Oriented Middleware* (*cMOM*). The behavior of cMOM is modeled through the concept of event-service interacting model. The model for cMOM specifies how events are handled, how reactions are triggered and executed - with respect to the applications that produce the events and that may be concerned by the execution of the reactions. The model consists of several parts: event specification language, *Event-Service-*

Event rule (ESE-rule) and its graphic representation, Event-Service-Chain, event detection, event retrieval and delivery. In the following, we introduce those aspects respectively.

3.1 Event Specification Language

In general MOMs, only primitive event is supported, it is not easy to tie together applications that have predefined events. Event composition obviously provides a flexible way to define various complex events. cMOM provides a means for specifying composite events in. For the definition of events in cMOM, an event specification language [9] is proposed. Different with other languages [10][11][12][13], this language incorporates a small number of event operators and allows users to add new composite events to cMOM. With the help of the event specification language, cMOM provides a richer set of event expression and a set of useful constructs for building composite events.

In this language, an event consists of two parts, the event descriptor and the event body. The event descriptor tells event mediator what to do with the message. It contains information such as the target and source of the event, the global identification of event, event name, timestamp, the identity of interaction group and the priority of the event. The event body is completely user customizable and contains the data to be sent between applications.

3.2 Event-Service-Event (ESE) and Event-Service-Chain (ESC)

The use of the reactive paradigm requires different execution models in a variety of contexts and applications. The behavior of cMOM is modeled through the concept of event-service interacting model. The main parts of model are *Event-Service-Event* rule (ESE-rule) and its graphic representation, Event-Service-Chain. ESE-rule is used to describe the relationship between events and services. The rules of Event-Service-Event are derived from *Event-Condition-Action-Alternative Action* (ECAA) [14]. The syntax of ESE-rules is given as follows:

< ESE-rule>::=on <Event> do <Service> produce <Event>
<Composite_Event>::=<Event>{{AND|XOR|OR}<Event>}*
<Event>::=<Composite_Event>|< Primitive_Event>
<Atomic_Service>::=<Operation>
<Complex_Service>::=<Service>{{AND|XOR|OR}<Service>}*
<Service>::=<Complex_Service>|< Atomic_Service>

According to ESE rules, each reaction is associated with an event type and produces another event type. When the reaction is triggered by an occurrence of an event which application registers interest in, the reaction outputs a new event occurrence, and the new event occurrence fires a new reaction. Therefore an event occurrence can lead to chains of reaction.

ESE-rule is employed to describe how applications interact. However, since ESE-rules are isolated, it is not straightforward. Thus, we adopt a graphic style to describe static interactions among distributed applications. The graphical representation is named as *Event-Service Graph* (ESG). ESG is a directed graph and is constructed from ESE-rules. If the input event of ESE rule is output event of another ESE rule, two rules are connected in ESG. Figure 1 gives an example of an ESG. In the figure, diamond, circle and rounded rectangle indicate event, logical connector, service or action respectively.

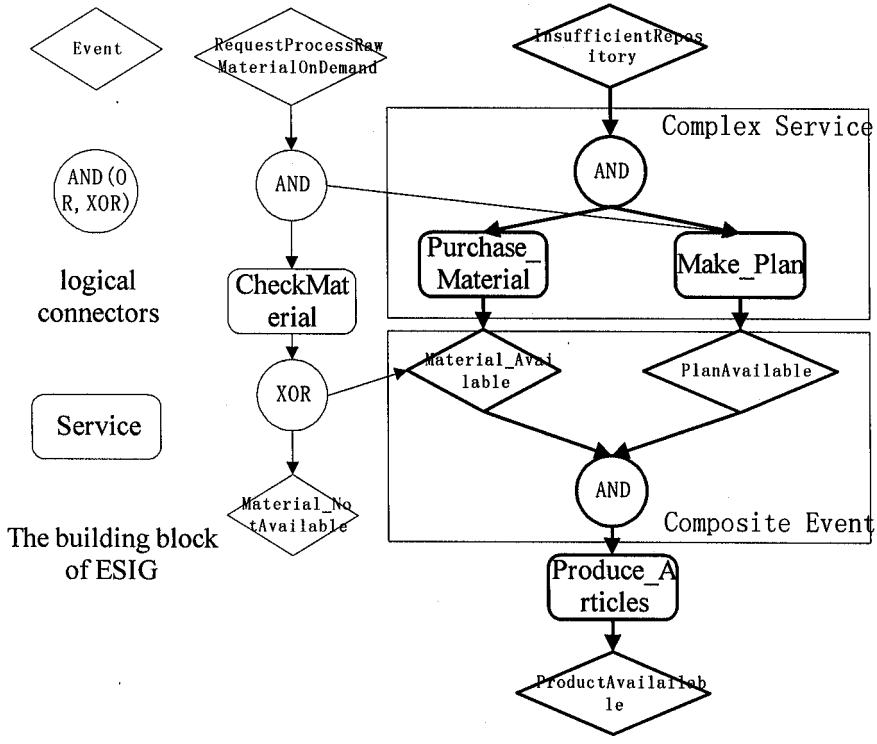

Fig. 1. A Simple ESG Example

Because ESG only represents possible interaction among services, the interactions occurring in reality are subset of ESG. Thus, *Event-Service-Chain* (ESC), constructed from ESG, is used to model dynamic interaction of applications. When one interaction is invoked, a corresponding ESC is created. When one invoking ends, it produces new event occurrences and those new event occurrences trigger other execution. With progress of the interaction among applications or services, the ESC is extending. Thus ESC keeps track of interaction. An example of ESC is shown in block line in Fig. 1. When event *InsufficientRepository* in Fig.1 occurs, the ESC indicated in block line is created.

ESC is not only used to track the execution of applications, but also detect event occurrence. Events occurring in an ESC may not combine with other ESCs, thus, it assures the correction of event semantics (The method to detect event occurrences will be discussed in section 3.3). Moreover, since events are transmitted through

cMOM, cMOM keeps track of the sequence of executing tasks and events by the execution model. Once exception has been detected, recovery invoked and recovery manager performs compensation services or tasks. If any compensation services or tasks exist, traces back the method execution log for semantic recovery.

3.3 Event Detection

Since an event may have multiple instances during a computation, composite event detection is more difficult than primitive event detection. Snoop [10] and TriGS [15] use syntax trees and syntax graphs. SAMOS [16] uses colored Petri nets for the detection of composite events. Ode [17] uses finite state automata, which are based on the equivalence of event expressions and regular expressions. The way of event detection mentioned in [10][15][16][17] did not consider event context, it often incurs semantic ambiguity, thus limiting the expressiveness of those general event specification languages.

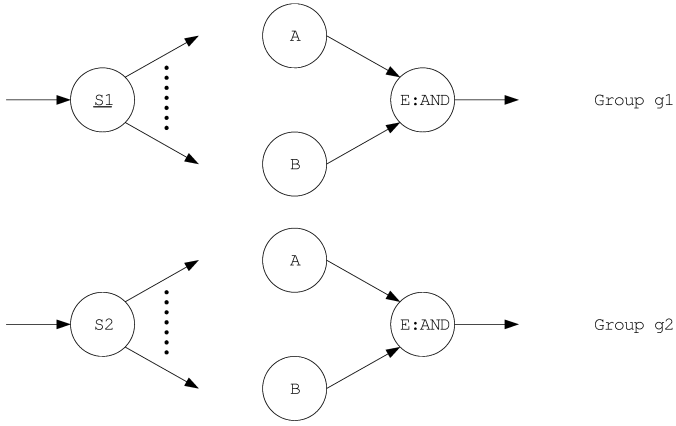

Fig. 2. An Example of Two Interacting Groups

cMOM implements an efficient composite event detection method. In cMOM, the ESC is the context of all events occurrences. The ESC is named as interacting group and assigned a group identifier - *GroupId* by cMOM. When an interaction is fired, an interacting group is created, and a group identifier is assigned. During the interaction, every event occurrence is assigned the group identifier. Events with different group identifiers do not form new composite events. An example shown in Figure 2 will help to understand the method. Fig.2 illustrates two interacting groups g_1, g_2, where events *A* and *B* are generated and a composite event *E* is also generated. A composite event *E* defined with AND operator and its component events are event *A* and *B*. When interaction reaches the nodes *A* and *B* of groups g_1, g_2, there will be four event instance occurrences a_1, a_2, b_1, b_2 of *A* and *B* (we use a small letter with a subscript index as the event identifier), where a_1.*GroupId*=b_2.*GroupId* and a_2.*GroupId*= b_1.*GroupId*. So event occurrences a_1 and b_2, a_2 and b_1 form two occurrences of event E. Other potential triggering combinations of events, such as a_1 and b_1, a_2 and b_2 are not tested, because the occurrence of *E* caused by (a_1, b_1) or (a_2, b_2) is not possible.

3.4 Event Retrieval and Delivery

In most MOMs, timely detection and delivery of various events are usually implicitly required but seldom explicitly defined; let alone any effort to guarantee it. Simply queuing events and dispatching them in a best-effort approach at the MOM does not guarantee the timely delivery of these events. Formalism for specifying various event-timing requirements and corresponding dispatch algorithms are needed.

When event occurrences are received by cMOM, cMOM inserts those event occurrences into *Handling Event Queue* (HEQ). Event processor retrieves event occurrences in HEQ. There are several ways in which an event can be retrieved from a queue. If an event is more important, it can be manipulated in several ways rapidly. For example, an application can use message priority levels for cMOM to deliver urgent messages first. The priority level of event occurrence ranges from 0 (lowest) to 9 (highest). If application does not specify a priority level, the default level is 4. cMOM tries to process higher-priority messages before lower-priority ones. If the event occurrences have the same priority level, event occurrences are processed according to expiration time. cMOM tries to process event occurrences before event occurrences expire.

4 Framework Implementation

cMOM requires four basic types of event service: delivery, receive, storage and process. The delivery service distributes events as they are processed. The storage service stores events for later retrieval. In addition, cMOM provides applications with service to register and un-register their ESE-rules. cMOM needs to be augmented for composite event specification and detection, as well as timely delivery semantics which are necessary in many real world applications. We first describe the requirements for these services.

- Transparency. Brokers should not learn about which brokers produce the events they consume, or which brokers consume the events they produce.
- Specificity. A broker should be passed only the set of events it currently expresses an interest in. cMOM must match announced events to those brokers that have expressed interest in them.
- Dynamic. Brokers can register and un-register ESE-rules dynamically.
- Support for *quality of service* (QoS). In addition to end-to-end latencies and persistence, applications often have particular requirements for delivery ordering semantics during synchronous working.
- Event storage and retrieval. Users often need to go back in time selectively to review, replay and reconcile object histories.

We implement a framework of cMOM, which is composed of ES (Event Sender), ER (Event Receiver), EA (Event Analyzer), ET (Event synthesizer), EM (Event Matcher), ESE Register, and Interaction History Management. Figure 3 shows the overall architecture of cMOM designed to meet these requirements. As their names indicate, ES and ER send and receive events, respectively. Event producers and consumers transfer events over an interaction group, which manages event scoping as

well as storage. Events pertaining to a cluster of objects are propagated with a default scope that is equal to the collection of sites where those objects reside. Fig.4 also shows an import-export service, which is used for federating sessions, and an event store, which is used for temporary and permanent storage of events.

cMOM supports applications by providing event management, storage, and notification functionality. The middleware performs the function of collecting messages from producers, filtering and transforming them as necessary, and routing them to the appropriate consumers.

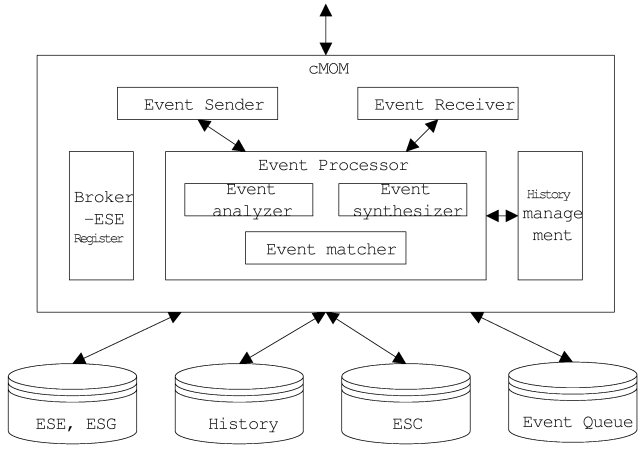

Fig. 3. The Architecture of cMOM

The principle of using cMOM is as follows: applications' ESE-rules are translated to rules stored and executed in cMOM. cMOM then performs event detection and determines applications that have registered a notification interest for such event occurrences. These applications are then appropriately informed and react as defined by their ESE-rules. The whole procedure of interaction is recorded by ESC. When a broker generates an event, it notifies the ER about the occurrence. The ER inserts the event into HEQ. ET retrieve event from HEQ, checks whether new composite events can be detected. EM determines brokers that have registered a notification interest for such event occurrences, and then adds the pair of action and event into SEQ (Sending Event Queue). The EA decomposes composite events into primitive events. The ES posts events to respective brokers.

5 Conclusion

In this paper, a composite-event-based Message-Oriented Middleware is developed. cMOM enables the integration of services and resources within and across enterprises or other organizations. We emphasize that the semantics is a key to capture the actual meaning of events in real life. Same composite event patterns may imply totally different semantic meanings in different context. We study an event and service interacting model. Based on this model, a graph-ESG, constructed according to ESE-

Rules, and a graph-ESC, which keep track of applications' interaction, are adopted to detect event. The ESG and ESC greatly enhanced the expressive power to detect event correctly, which matches situations in reality. The composite event detection method solves the problems that multiple event occurrences incur semantic ambiguity.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2] OMG, Event Service Specification, http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm.

[3] OMG, Notification Service Specification, http://www.omg.org/technology/documents /corbaservices_spec_ catalog.htm.

[4] IBM, MQSeries for Windows NT and Windows 2000 Version 5 Release 2.1, http://www-3.ibm.com/software/integration/wmq/v521/MQNT20002Q01521.pdf.

[5] M. Korhonen, Message Oriented Middleware (MOM), http://www.tml.hut.fi/Opinnot/Tik-110.551/1997/mqs.htm.

[6] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman, "An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems", *Proceedings of IEEE International Conference on Distributed Computing Systems*, Austin, TX, 1999.

[7] G. Banavar, M. Kaplan, K. Shaw, R. E. Strom, D. C. Sturman, and W. Tao, "Information Flow Based Event Distribution Middleware", *Proceedings Middleware Workshop at the International Conference on Distributed Computing Systems*, 1999.

[8] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra, "Matching Events in a Content-based Subscription System", *Proceedings of ACM Symposium on Principles of Distributed Computing*, Atlanta, GA, 1999.

[9] P. Yuan, G. Chen, J. Dong, and W. Han, "Research on an Event Specification for Event-Based Collaboration Support Software Architecture", *Proceedings of the Seventh International Conference on CSCW in Design*, Sep. 2002, pp.99-104.

[10] S. Chakravarthy and D. Mishra, "Snoop: An Expressive Event Specification Language for Active Databases", *Data and Knowledge Engineering*, 14(10), 1994, pp.1-26.

[11] N. Gehani, H. V. Jagadish, and O. Shmueli, "COMPOSE: A System for Composite Event Specification and Detection", *Advanced Database Concepts and Research Issues* (N. R. Adam and B. Bhargava, eds.), Lecture Notes in Computer Science, Springer Verlag, 1994.

[12] G. Liu, A. K. Mok, and P. Konana, "A Unified Approach for Specifying Timing Constraints and Composite Events in Active Real-Time Database Systems", *Proceedings of Real-Time Technology and Applications Symposium*, June 1998.

[13] S. Gatziu, K. R. Dittrich, **"**Detecting Composite Events in Active Database Systems Using Petri Nets", *Proc. of 4th Intl. Workshop on Research Issues in Data Engineering*, Houston, USA, 1994, pp.2-9.

[14] D. McCarthy and U. Dayal, "The architecture of an active database management system", *Proceedings of the 1989 ACM SIGMOD international conference on Management of Data*, Portland, Oregon, May-June 1989, Vol.18, No.2, 1989, pp.215-224.

[15] W. Retschitzegger, "Composite Event Management in TriGS Concepts and Implementation", *DEXA'98,* pp.1-15.

[16] S. Gatziu and A. Vaduva, "SAMOS: an Active Object-Oriented Database System: Manual", *Technical Report 96.02*, Computer Science Department, Univ. of Zurich, 1996.

[17] N. Gehani and H. V. Jagadish, "Ode as an Active Database System: constraints and Triggers", *Proc. of the International Conference on VLDB*, September 1991, pp.327-336.