한국산학기술학회논문지
Vol. 11, No. 6, pp. 2256-2262, 2010

Business Process Monitoring under Extended-GMA Environment with Complex Event Handling

Minsoo Kim[1] and Young Seok Ock[1*]
[1]**Department of Systems Management and Engineering, Pukyong National University**

확장된 GMA 환경 하에서 복합 이벤트 처리를 통한 비즈니스 프로세스의 모니터링

김민수[1], 옥영석[1*]
[1]부경대학교 시스템경영공학과

Abstract The requirements for automated handing of business process and its monitoring usually have a proprietary form for each enterprise. Unlike the conventional database transaction, business process takes long time for its completion and incorporates very complex handling logics along with business situations. Since those handling logics are frequently changing in accordance with the business policies or environment, enterprises want to integrally capture the whole business semantics while monitoring those process instances. In this paper, we adopted GMA(Grid Monitoring Architecture) for the integrated monitoring of business processes. The GMA(Grid Monitoring Architecture) is a very scalable architecture to effectively monitor and manage monitoring information under the heterogeneous environment. By introducing complex event handling features into GMA to support various processing logics, we could implement a system that enables automated execution and high-level monitoring of business processes.

요 약 비즈니스 프로세스의 자동화된 처리와 이에 대한 모니터링 요구는 기업마다 특화된 형태를 지니는 것이 일반적이다. 비즈니스 프로세스는 일반적인 데이터베이스 트랜잭션과는 달리 수행에 장시간이 소요되며, 비즈니스 상황에 따라 매우 복잡한 처리 로직을 가진다. 이러한 처리 로직들은 사업 정책이나 환경이 변함에 따라 자주 변경되기 때문에 기업은 특정 비즈니스 프로세스의 모니터링 과정에서 해당 프로세스 인스턴스의 전체적 의미를 종합적으로 파악하고자 한다. 본 연구에서는 비즈니스 프로세스의 종합적인 모니터링을 위해 GMA(Grid Monitoring Architecture)를 사용하였다. GMA는 이종 시스템 환경 하에서 모니터링 정보를 효과적으로 관리하고 감독하기 위하여 제시된 확장성이 높은 아키텍처이다. 다양한 처리 로직을 지원할 수 있도록 GMA 모델에 복합 이벤트의 처리 기능을 부가함으로써, 비즈니스 프로세스에 대한 자동화된 실행과 상위수준의 모니터링이 가능한 시스템을 구축할 수 있었다.

Key Words : Grid Monitoring Architecture, Business Process Automation, Service Oriented Architecture, Complex Event Processing.

1. Introduction

It is generally accepted that the performance monitoring for business transaction system is one of the important software components in the enterprise environment. Since the performance degradation of enterprise system or even worse the system failure strictly affects enterprise's revenue, it is mandatory to monitor the system's performance in real time and response proactively before those problems[1, 2, 3]. However,

This work was supported by Pukyong National University Research Fund in 2007(0012000200710200)
*Corresponding Author : Ock, Young Seok(ysock@pknu.ac.kr)
Received May 11, 2010 Revised June 3, 2010 Accepted June 18, 2010

monitoring for business transactions is a very challenging task in the enterprise environment.

Unlike the database transaction that usually completes within a millisecond, business process takes long time for its completion, and incorporates very complex handling logics along with business cases and situations. Consider a typical order handling process. Once a quote is accepted by the customer who issued quote request to a company, corresponding order can be placed to the company, and subsequent manufacturing activities are scheduled. After the production of the ordered items, shipping for delivery is scheduled. This process also continues to the invoice and remittance handling. The whole order processing may take days or even weeks and sub-activities are semantically connected. The meaning of semantically connected indicates that sub-activities are connected not by hard-coded logics but by causal relationships. What is even worse with the order processing is that it can be revoked during the process under certain conditions. Even the completed order can be revoked, and causes compensation transaction to occur. In addition to this long running feature of semantically connected sub-activities, business transaction usually accompanies large volumes of data and human intervention. In case of order processing, item description and status report document are usually accompanied in a human-readable format. These documents are quite voluminous compared with the data volume in typical database transaction. Due to all those factors, handling logics for business process are very complex and hard to implements, and those handling logics are frequently changing in accordance with the business policies or environment.

With all above difficulties, monitoring for those business transactions becomes a challenging task and enterprises want a system that integrally captures the whole business semantics while monitoring those business process instances. To handle these monitoring issues, various architectures and technologies have been suggested. Among them, there is a Grid Monitoring Architecture (GMA) that is developed by Performance Working Group of Global Grid Forum (GGF) as an open architecture for technically heterogeneous system's monitoring[4, 5]. GMA has very simple and high-level monitoring structure that can cover wide variety of systems. Though this high-level structure can provide extensibility, on the other hand it lacks some significantly required factors for enterprise system's monitoring.

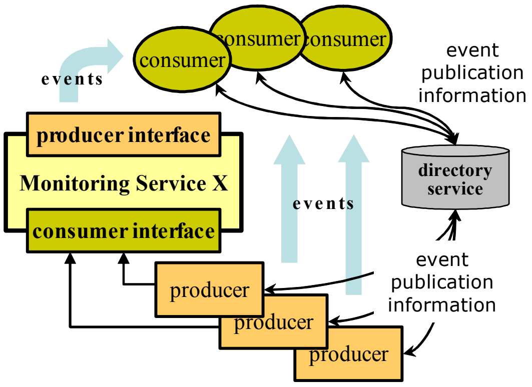
In this paper, we suggest an extended monitoring architecture that provides complex event handling over GMA to support business processes. This paper is composed as follows. In section 2, we introduce some previous researches. In section 3, an overview of GMA is given in detail with its limitations for business process monitoring. In section 4, our extension to GMA and its implementation is described. Finally, in section 5, we conclude our research.

2. Related Researches

2.1 Grid Monitoring Architecture

As is described in the previous section, GMA is a general monitoring and management architecture for high-performance distributed computing. Though this generality of GMA provides the architecture higher level of extensibility, it is often pointed out as a main cause of ambiguousness that makes it hard to implement the architecture for specific domains[5, 6].

The GMA consists of three types of components. Figure 1 shows these components with their interactions. A producer which is often referred to as performance event source is any component which sends events to a consumer. The opposite is a consumer which receives event data from a producer. Consumer is often called performance event sink. An event is a typed collection of data with a specific structure that is defined by an event



[Fig. 1] GMA components and service

schema. A directory service contains information about producers and consumers that accept requests. Producers publish their existence in a directory service and consumers can use this information to locate producers of interest. Once a consumer subscribes to a producer, then it can receive performance data from the producer[4, 5, 6].

A producer can use a directory service to discover consumers of interest as well. Either a producer or a consumer can initiate the interaction to a discovered peer. In either case, performance event data is always sent directly from a producer to a consumer with no further intervention of directory service[4, 5].

Besides above three components, GMA also suggests intermediary component to provide compound services and to shun a heavy workload for a producer. Intermediary is a compound component that is both consumer and producer. In the figure 1, 'Monitoring Service X' is an intermediary that combines two producers' events and provides derived performance events to other consumer. Other typical example of intermediary can be a broadcaster that relays a producer’s events to other consumers. Via these intermediaries, we can easily enrich monitoring services that operate on the GMA.

Many researches and projects have been done over GMA such as pyGMA, GrADS, AutoPilot and R-GMA. In those researches, several extensions to GMA have been applied to monitoring systems of various application domains including nuclear physics, network monitoring, and job accounting in virtual organization[5, 7-9].

2.2 Complex event handling

To support complex processing logics in a dynamic business process, a monitoring system that is capable of tracing business semantics over distributed environment is required. Among the distributed architectures, SOA (Service Oriented Architecture) and EDA(Event Driven Architecture) are highlighted nowadays. In the table 1, several features of SOA and EDA are compared with each other.

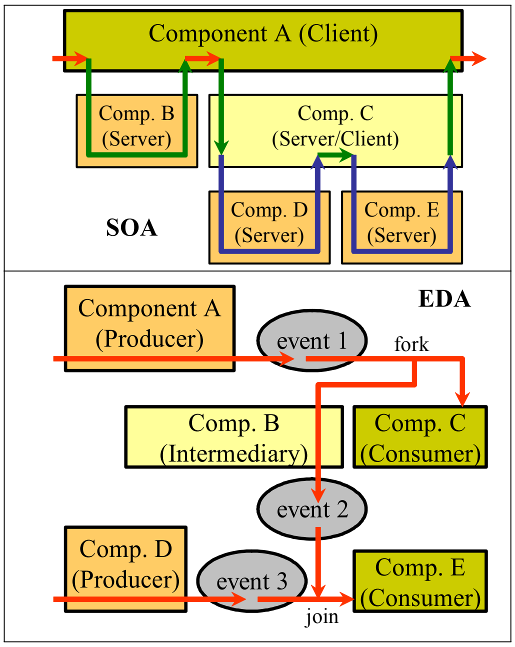
[Table 1] Features of SOA and EDA

	SOA	EDA
interaction	1:N	N:M
initiator	client(consumer)	receiver(producer)
addressing	direct/indirect	indirect
collaboration	pull	push
flow control	sequential	dynamic/parallel
communication	synchronous	asynchronous
coupling	loosely coupled	highly decoupled

Under SOA, each component describes its own services and access method via WSDL (Web Service Description Language) standard, and registers this description to a standardized directory called UDDI (Universal Description, Discovery and Integration). Once a user component finds target component via UDDI lookup, then it can call the services directly to that registered component[10, 11]. It can be said that the essence of SOA lies in the whole standardization of service description, registry and invocation. If SOA focuses on the sequential invocation of services on the component web, EDA focuses on the production, detection, and consumption of every event. An event can be defined as a significant change in the state of component. EDA, by design, allows applications to be more responsive to changes of state, and is generally believed to be more acceptable for asynchronous environments.

Figure 2 shows how the individual components are organized to complete a service transaction. In the SOA environment, this can be done by 1:1 service call between components that sequentially cooperate in a synchronized request-response way. On the other hand, in the EDA environment, each component asynchronously performs specific function according to the events generated. Events are triggered by component and can be notified to one or more components.

Components that are notified those events can subsequently trigger other events as well. As we can see in the figure 2, events can join together in the middle of propagation, or an event can be split into multiple events. Along these event propagation chains, service can be collectively completed by the functionalities of highly decoupled components. This kind of complex event handling is generally referred to as CEP (Complex Event Processing), and widely used in various application domains including grid and cloud computing[12-16].



[Fig. 2] Service composition in the SOA and EDA

Nowadays, EDSOA (Event-Driven SOA) or SOA 2.0 is widely discussed as a next generation architecture that enriches SOA with event handling feature.

3. Limitation of GMA

In fact, GMA introduces just a simple architectural design that performance data are generated by producer and are consumed by consumer. Performance data are consumed in a way of query/response, publish/subscribe and notification. GMA states consumption of events, but it does not clearly mention how they are really defined and processed. As is described earlier, event processing is a very crucial element in the monitoring of business process, so the exact mechanism to support event handling is mandatory to implement those systems over GMA.

From the perspective of system monitor, query/response method is an interface to get information of interest by providing necessary conditions. GMA, however, lacks specification of meta-data for describing the query. In order to specify query conditions, meta-data, which is called schema, is needed to describe queried data.

Publish/subscribe is an asynchronous approach to receive performance data by subscribing to producer in advance. This interface can be used for a monitoring application to continuously trace the changes in the system. By scattering an event to multiple subscribers, we can implement fork of an event with publish/subscribe interface.

In the notification, once a consumer registers its necessary condition to a producer, that producer notifies corresponding events to the consumer when the condition is met. In a sense, notification is a conditional publish/subscribe approach. Publish/subscribe and notification approaches are basically 1:1 asynchronous communication. However, we can extend this interaction to N:1. If we use intermediary to subscribe multiple producers then we can emulate N:1 interaction by attaching a new notification request to this intermediary.

With basic GMA interactions, EXCLUSIVE-OR branching is not possible, because the publish operation distributes an event to all subscribers. History-based processing of accumulated events is also a problem under GMA. If we want to fire a critical event for every five minor events, it seems very hard to be implemented. Extending those basic GMA interactions to support N:1 and N:M is somewhat cumbersome, so it is needed to extend basic GMA interfaces.

4. Extending GMA with CEP

As is depicted in the above section, there exist two limitations in the GMA. The first is that producers do not provide meta-data interface for the performance data which they generate. Description of condition for querying, subscribing and notifying events cannot be done without meta-data interface. The second is that with a simple pattern of 1:1 asynchronous interaction, it is very hard to correspond to diversifying monitoring requests. Thus, a more comprehensive and expanded architecture is required.

4.1 Meta-data interface

Providing meta-data interface can be done in two different ways. The first approach is to register meta-data together with producer's event publication information when a producer registers its event publication information to the directory service. The other approach is to provide meta-data query service in the producer interface.

The former approach that consults directory service for meta-data whenever a performance data is required can degrade the performance of the directory service. Workload on the directory service can be enormously increased as the number of producer grows, and thus can bring about service latency. In that sense, the latter approach is practically more preferable.

We added ‘Get Performance metadata’ service to the producer’s basic 8 services which are

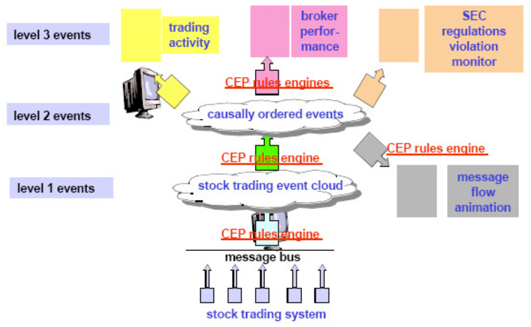
‘Authorize from consumer’,
‘Authorize to consumer’,
‘Query’,
‘Consumer-initiated Subscribe’,
‘Consumer-initiated Unsubscribe’,
‘Producer-initiated Subscribe’,
‘Producer-initiated Unsubscribe’,
and ‘Version’.

It is desirable to standardize the description method of performance meta-data just like WSDL in the SOA. In this paper, we chose WSDL to describe performance meta-data. Of course, it can be vary based on the user’s request.

4.2 Extended interaction with CEP

To overcome the simple event handling mechanism of GMA, applying the CEP concept of EDA which supports various events handling mechanism is required.

In the CEP, all the events of enterprise system are generated in either the middleware layer, application layer or business process layer. Once an event is issued, lower level event can affect to the occurrence of higher level event. The relationships between low level and high level event can be causal, sequential, aggregation or so.



[Fig. 3] Multi-level events in the stock trading system[12]

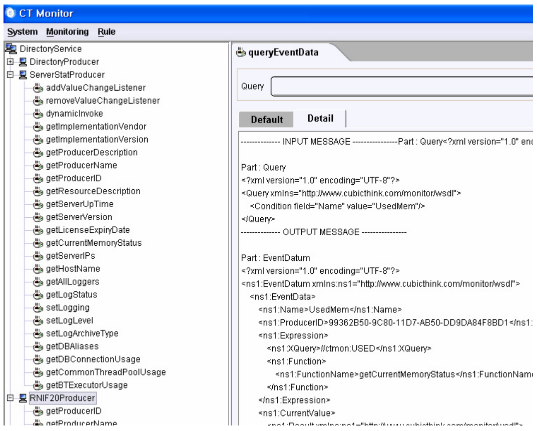
David Luckham who first introduced CEP concept suggested Event Processing Agent (EPA) as a software module that connects the generation of low level event to that of high level event[12]. Actually EPA can be made with a rule engine that detects the occurrence of pre-described condition by analyzing the patterns of low level events, and thus triggers high level event based on that condition. By introducing a rule engine to GMA, we can provide more complex interaction patterns with complex event processing.

In our extended GMA, CEP rule engine is an intermediary component. It is a consumer from a viewpoint that it processes low level event of the producer, and it is a producer from a viewpoint that it generates high level event by combining multiple low level events.

Deploying CEP rule engine as an intermediary component has several pending issues. One issue is related with the problem of how to synchronize event definition between multiple rule engines. Since event definitions are dynamically changing on the fly, propagating and synchronizing those changes across multiple rule engines are important to guarantee exact operation of extended GMA. In our implementation, we introduced only one CEP rule engine to shun those synchronization issues.

4.3 Pilot implementation for business process

To demonstrate the feasibility of our extended GMA, we implemented a monitoring system for business process. Our pilot system is to trace RosettaNet B2B transactions between trading companies. Figure 4 shows our monitoring system's service description screen.



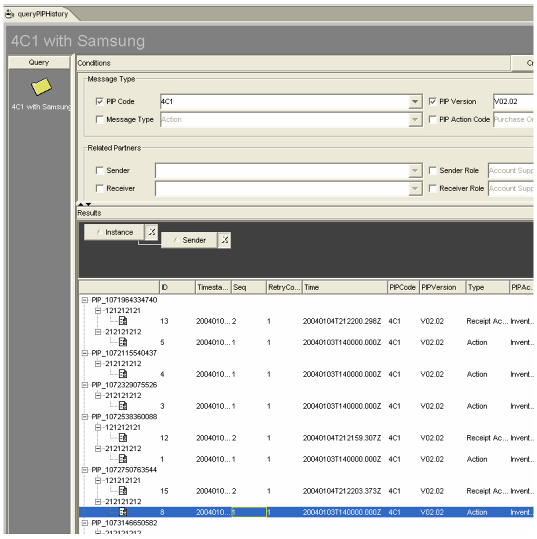
DirectoryService
DirectoryProducer
ServerStatProducer
addValueChangeListener
removeValueChangeListener
dynamicInvoke
getImplementationVendor
getImplementationVersion
getProducerDescription
getProducerName
getProducerID
getResourceDescription
getServerUpTime
getServerVersion
getLicenseExpiryDate
getCurrentMemoryStatus
getServerIPs
getHostName
getAllLoggers
getLogStatus
setLogging
setLogLevel
setLogArchiveType
getDBAliases
getDBConnectionUsage
getCommonThreadPoolUsage
getBTExecutorUsage
RNIF20Producer
getProducerID
getProducerName

```
------------- INPUT MESSAGE ---------------Part : Query<?xml version="1.0" enc

Part : Query
<?xml version="1.0" encoding="UTF-8"?>
<Query xmlns="http://www.cubicthink.com/monitor/wsdl">
  <Condition field="Name" value="UsedMem"/>
</Query>
------------- OUTPUT MESSAGE ---------------

Part : EventDatum
<?xml version="1.0" encoding="UTF-8"?>
<ns1:EventDatum xmlns:ns1="http://www.cubicthink.com/monitor/wsdl">
  <ns1:EventData>
    <ns1:Name>UsedMem</ns1:Name>
    <ns1:ProducerID>99362B50-9C80-11D7-AB50-DD9DA84F8BD1</ns1:
    <ns1:Expression>
      <ns1:XQuery>//ctmon:USED</ns1:XQuery>
      <ns1:Function>
        <ns1:FunctionName>getCurrentMemoryStatus</ns1:FunctionNam
      </ns1:Function>
    </ns1:Expression>
    <ns1:CurrentValue>
```

[Fig. 4] Monitoring service description

In the left pane of figure 4, a list of services that are registered to UDDI is shown. In the right pane, selected service interface's WSDL detail is given in XML format

Figure 5 shows sample screen for event composition. We provide drag&drop feature to combine events easily.



PIP Code: 4C1 [✓]
Message Type: Action
PIP Version: V02.02 [✓]
PIP Action Code: Purchase Or
Sender:
Sender Role: Account Supp
Receiver:
Receiver Role: Account Supp

	ID	Timesta...	Seq	RetryCo...	Time	PIPCode	PIPVersion	Type	PIPAc...
PIP_1071964334740									
121212121									
	13	2004010...	2	1	20040104T212200.298Z	4C1	V02.02	Receipt Ac...	Invent...
212121212									
	5	2004010...	1	1	20040103T140000.000Z	4C1	V02.02	Action	Invent...
PIP_1072115540437									
212121212									
	4	2004010...	1	1	20040103T140000.000Z	4C1	V02.02	Action	Invent...
PIP_1072329075526									
212121212									
	3	2004010...	1	1	20040103T140000.000Z	4C1	V02.02	Action	Invent...
PIP_1072538360088									
121212121									
	12	2004010...	2	1	20040104T212159.307Z	4C1	V02.02	Receipt Ac...	Invent...
212121212									
	1	2004010...	1	1	20040103T140000.000Z	4C1	V02.02	Action	Invent...
PIP_1072750763544									
121212121									
	15	2004010...	2	1	20040104T212203.373Z	4C1	V02.02	Receipt Ac...	Invent...
212121212									
	8	2004010...	1	1	20040103T140000.000Z	4C1	V02.02	Action	Invent...
PIP_1073146650582									

[Fig. 5] Event composition for business monitoring

5. Conclusion

In this paper, we supplemented the simple event handling mechanism of GMA by incorporating CEP concept of EDA to support the monitoring of business process. We also suggested the performance meta-data interface to effectively support query description.

Many trials have been carried out to extend GMA. Most of those efforts are somewhat related to the implementation of GMA for complex event support. For those researches, our research will be helpful as a reference implementation.

참고문헌

[1] F. Casati et al., "Specification and implementation of exceptions in workflow management systems", *ACM Transactions on Database Systems*, Vol.24, No.3, pp.405-451, 1999.

[2] W.M.P. Van Der Aalst et al., "Case handling: A new paradigm for business process support", *Data & Knowledge Engineering*, Vol.53, No.2, pp.129-162, 2005.

[3] M. Kim and D. Kim, "FAULT-TOLERANT PROCESS DEBUGGER FOR BUSINESS PROCESS DESIGN", *International Journal of Innovative Computing, Information and Control (IJICIC)*, Vol.6, No.4, pp.1679-1687, April, 2010.

[4] R. Aydt, et. al., "A Grid Monitoring Architecture", *http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf*, July, 2001.

[5] A. J. Wilson et al., "Information and Monitoring Services within a Grid Environment", *Journal of Grid Computing*, Vol.2, No.4, December, 2004.

[6] M. Kim and Y. S. Ock, "Extending a Grid Monitoring Architecture for Complex Event Handling", *IT Promotion in Asia (ITPA)*, pp. 94-98, September, 2009.

[7] F. Berman et al., "The GrADS Project: Software Support for High-Level Grid Application Development", *International Journal of High Performance Computing Applications*, Vol.15, No.4, pp.327-344, 2001.

[8] W. Allcock et al., "The Globus Striped GridFTP Framework and Server", *Proceedings of Super Computing 2005*, November, 2005.

[9] A. W. cooke et al., "The Relational Grid Monitoring Architecture: Mediating Information about the Grid", Journal of Grid Computing, Vol.2, No.4, pp.323-339, December, 2004.

[10] T. Hugh et al., "Event-Driven Architecture: How SOA Enables the Real-Time Enterprise",

Addison-Wesley Professional, 2009.

[11] E. Thomas, "Service-Oriented Architecture: Concepts, Technology, and Design", Prentice Hall, 2005.

[12] D. Luckham, "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems", *Addison-Wesley Professional*, 2002.

[13] K. Czajkowski et al., "Grid Information Services for Distributed Resource Sharing", *10th IEEE International Symposium on High Performance Distributed Computing,* pp.181-184, IEEE Press, New York, 2001.

[14] B. Fran et al., "Grid Computing: Making the Global Infrastructure a Reality", *Wiley*, 2003.

[15] 강윤희, 강경우, 조광문, 김도현, 궁상환, "P2P를 기반으로 한 확장된 그리드 정보 서비스 시스템", 한국산학기술학회논문지, v.4, no.3, pp.270-273, 2003년 9월

[16] 국유근, 이준, 김재수 "정보 상호운용을 위한 전자정부 그리드 시스템", 한국산학기술학회논문지, v.10, no.12, pp.3660-3667, 2009년 102월

Minsoo Kim [Regular member]

- Feb. 1996 : Seoul National Univ., Industrial Engineering, MS
- Aug. 2002 : Seoul National Univ., Industrial Engineering, PhD
- Oct. 2004 ~ current : Pukyong National Univ., Dept. of Systems Management and Engineering, Assistant Professor

<Research Interests>
Manufacturing Information System, e-Business, BPM

Young Seok Ock [Regular member]

- Feb. 1984 : KAIST, Industrial Engineering, MS
- Feb. 1993 : KAIST, Industrial Engineering, PhD
- Mar. 1987 ~ current : Pukyong National Univ., Dept. of Systems Management and Engineering, Professor

<Research Interests>
Manufacturing Information System, Distributed Resource Information Management