JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 1073-1089 (2011)

Implementation of Business Logic Framework for Collaboration of Heterogeneous Service Systems in RFID/USN Environment

SEOK-KYOO KIM[1], JUNO CHANG[2] AND SANG-YONG HAN[3]
[1]*Infrastructure Technology Development*
Korea Institute of Science and Technology Information
Daejeon, 305-806 Korea
[2]*Division of Digital Media Technology*
Sangmyung University
Seoul, 110-743 Korea
[3]*School of Computer Science and Engineering*
Seoul National University
Seoul, 151-744 Korea

Currently the research and development of various types of middleware for the ubiquitous computing environment are in demand, and many application services are therefore being provided. The practical ubiquitous computing environment is world in which a variety of services based on RFID/USN devices are integrated, creating synergy effects. From perspective of the end user, there should be a middleware platform and a business logic framework that join various services together to satisfy a range of user requirements. In addition, the platform should provide the user with a description of a scenario based on various events to maximize the collaboration. In this paper, we propose a business logic framework that supports collaboration between heterogeneous service systems based on user scenarios. We also attempt to show the efficiency and scalability of the proposed framework by providing the result of several tests.

Keywords: ubiquitous computing environment, middleware platform, business logic framework, scenario-based, RFID/USN

1. INTRODUCTION

As advancements in computer and network technologies continue, the ubiquitous computing environment with various application services becomes more developed. Interest in making organic connections or integrating existing independent application services have been increasing as well. In particular, as a measure to realize ubiquitous computing in both a comfortable setting and in industrial environments, services based on a Ubiquitous Sensor Network (hereafter referred to as a USN) have been proposed. Connecting these services now enables composite integrated services.

To provide complex services for interconnected applications in a RFID/USN environment, an integrated USN application service infrastructure is necessary to support the reliable collection of a large amount of sensor data and real-time delivery of that data according to a predefined scenario for each service. As a solution for this requirement, a scenario-based business logic framework is implemented for cooperative work among heterogeneous service systems in the RFID/USN environment.

Received August 20, 2009; revised August 3, 2010; accepted October 5, 2010.
Communicated by Jonathan Lee.

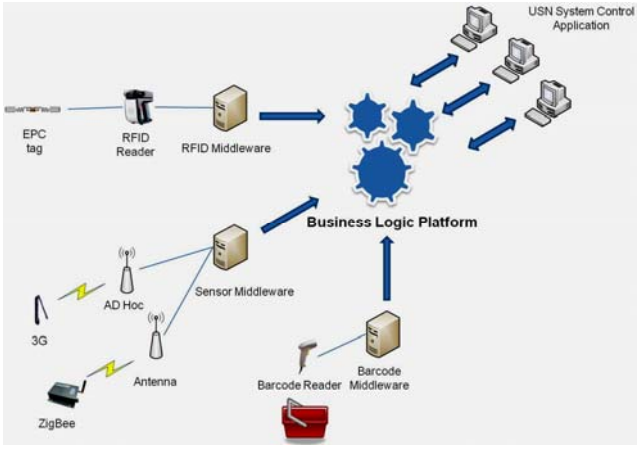


Fig. 1. Scenario-based collaborative business logic framework for heterogeneous service systems in a RFID/USN environment.

The business logic framework is placed between USN middleware, generally composed of RFID (Radio Frequency Identification) or sensor middleware and business-specific application services. The framework collects well-formed reports from the USN middleware and processes them by interpreting and delivering them to the corresponding user-defined business processes for application services in order to establish an integrated cooperative working environment.

The business logic framework adopts the service-oriented architecture (SOA) as an interfacing scheme for integrating RFID/USN-based application services not only to guarantee independence of individual services, but also to gain flexibility when extending the platform infrastructure by enabling new collaborative services to work with the platform or other application services. This research also plays an important role as an interface to the previously designed MESSA (Massive Events Service System Architecture) system [9, 10], which is prototyped to provide effective data stream management in the RFID/USN environment, as well as to external business application services. The SOA essentially works through the following steps: existing application functions are classified into functional business units; they are re-combined as software component units, namely, services, through standardized calling interfaces; and the services are combined to construct real-world business applications.

The property of the SOA known as “write once, reuse many times” makes it possible to increase the reuse of the business services. Easily applicable service functionalities through the SOA also provide the foundation for the stable provision of composite services. An additional advantage of using the SOA is that it retains the loose coupling of the service building blocks across the platform infrastructure. Lowering the degree of coupling by interfacing the components is one of the most important objectives of software development methodologies, along with an increase in reuse. Another important reason for the platform to adopt SOA technology is that it allows a business application to request any services that are exposed to others with SOA interfaces regardless of the base platform of the services. Moreover, the SOA is considered to be a flexible connection structure that exists in parallel with both the rapid development of RFID/USN computing tech-

nology and the sharp increase in the demand for the related device-oriented services [8-10].

The proposed framework is based on the event-driven architecture (EDA), which is completely decoupled from the USN middleware. This scheme enables the platform to work independently with a specific USN middleware and execute scenarios only when the relevant events from the USN middleware are recognized. Based on the SOA structure, the platform exposes customized web services as a method for establishing connections with heterogeneous external systems. To promote the portability of the business logic framework, it is written in the Java programming language. In addition, a script language referred to as "XLogic (eXtensible Logic)" allows users to register the scenarios that are activated by a specific event and deliver the event to related service applications. In order to support real-time processing, a caching system is used. This system keeps the XLogic scripts in the memory in a manner that depends on their frequency of use. This scheme eventually shortens the time for parsing the scripts.

This paper is composed of 6 sections in total. Sections 1 and 2 correspondingly present the introduction and the related research. Section 3 describes the organization and functionalities of the business logic framework – the result of the present research. In section 4, a test scenario to test the constituent functions of the business logic framework is presented, and the result of its execution is described. The performance test is summarized in section 5, and section 6 concludes with the future research directions.

2. RELATED RESEARCH

2.1 USN Middleware

The requirements of the early types of USN middleware were comparatively simple due to the typically direct connection to a specified application service. As various types of application services, such as u-Healthcare, u-Transportation, u-911, and u-Eco based on a ubiquitous environment are rapidly introduced, however, it is necessary that the information from many RFID/USN devices interrelates with each other [2].

Other middleware research is being conducted on the following examples: MiLAN middleware [1], developed to guarantee QoS (Quality of Service) as its top priority; event-based DSWare middleware [5] , which sends desired data to USN application systems by setting events on continuous stream of sensor data as it is obtained, similar to RFID middleware; Impala middleware [6], which can dynamically change the functions of sensor node middleware according to changes in USN application services and changes in the surrounding environment of sensor networks through wireless communication; TinyDB middleware [7] and Cougar middleware [11], which carry out the requirements of USN application systems using distributed processing by considering the sensor data from the sensor network as distributed data of a distributed database.

2.2 COSMOS

In COSMOS (COmmon System for Middleware Of Sensor network), developed by ETRI, the key functions of middleware, jointly needed for various types of USN application services, are extracted, and technology development and standardization to provide

these in a standardized way are carried out [3, 4]. The main functions of COSMOS are as follows: query support (temporary, permanent, and event-related): support for the simultaneous processing of a large amount of queries for large-capacity sensor network environments; and support for the abstraction of heterogeneous sensor networks.

COSMOS is broadly divided into three layers in its performance of the above functions. First, there is the sensor network abstraction layer, the part that actually makes the connection with the sensor network infrastructure, which enables easy integration of heterogeneous sensor networks and monitors the status of the sensor network in real time. Second, there is the sensor network intellectualization layer, the stage where meaning is given to sensor data as it is entered continuously. This layer manages sensor data efficiently and creates new context data by analyzing the sensor data and pre-established business data. Lastly, there is the USN service platform layer, the stage that efficiently supports the development of USN application service systems [3, 4]. It is responsible for the following tasks: providing open API to simplify the use of USN middleware; managing users who make use of multiple services; supporting connections with external service systems; and providing in real-time static/dynamic meta-data related to sensor nodes and sensor networks, which USN application service systems require.

2.3 Event-Driven SOA

In July of 2006, the Gartner Group introduced the concept of SOA 2.0, which combined EDA (Event-Driven Architecture) and SOA, thus representing the future development direction of SOA [12]. The concept of SOA 2.0 can be characterized as a combination of EDA and SOA. It is also known as event-driven SOA. If SOA is a model in which services are connected through requests and replies using a predefined service interface, EDA is a model in which events are sensed and processed in real-time (Table 1).

Table 1. EDA vs. SOA.

	EDA	SOA
Trigger	Event-driven Trigger	Service Consumer
Protocol Information	Event Standard Information	Service Interface Information
Flow Control Party	Event Receiver	Client
Connection Scheme for Communication/Interaction	Dynamic, Parallel, Asynchronous One-way Decoupled 1:1, 1:N, N:N	Sequential, Synchronous Two-way(Request/Response) Loosely-Coupled 1:1
Main Usage	Real Time Service	Service Reuse and Sharing

As a result of a combination of the EDA and SOA schemes, SOA 2.0 takes full advantages of both systems. A system based on SOA 2.0 is capable of sensing sudden changes in the environment and actively reflecting the changes to the system by modifying the inter-relationships between related service systems. This scheme encourages the service systems to provide pertinent services and enables efficient process service requests and system connections. Various systems based on SOA 2.0 are being developed by Oracle [13], BEA [14], and TIBCO [15].

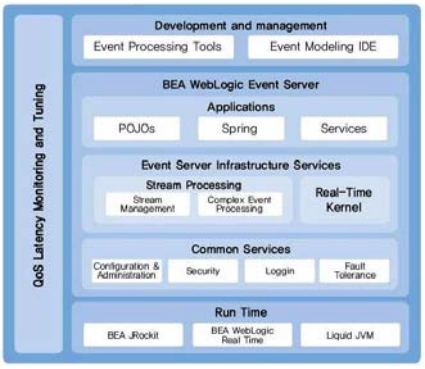


Fig. 2. BEA event-driven SOA.

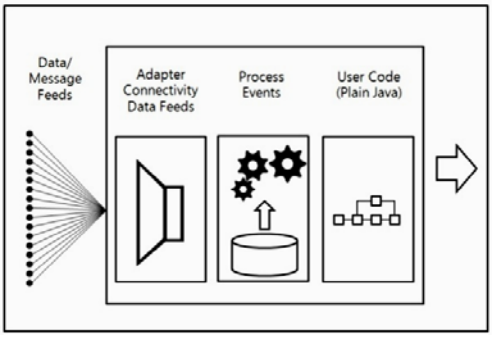


Fig. 3. BEA event-driven SOA solution's event processing stage.

2.4 BEA Event-Driven SOA Solution

BEA Event-Driven SOA is a solution based on a new software paradigm that can process various large-quantity incoming events. Using this scheme, companies can effectively predict future events to maximize their ROI [14]. A block diagram of BEA Event-Driven SOA is shown in Fig. 2.

BEA Event-Driven SOA is composed of three major products. The first is the WebLogic Event Server, which is a Java container for large-capacity, real-time, event-driven applications. The second is known as WebLogic RealTime, which is an application server that supports applications requiring a short waiting times as well as fast and predicable reply times in the standard Java-based infrastructure. The last component is the WebLogic RFID Edge Server, which controls RFID readers and other devices and supports event data stabilization and filtering. It is in charge of delivering data to the WebLogic Event Server. The processing order, from the event-delivery stage, is shown in Fig. 3.

The business logic framework proposed in this paper can make cooperative work possible among existing middleware systems and enable a variety of RFID/USN-enabled services to be combined within a flexible framework to provide composite and refined services. However, most middleware platforms and business logic frameworks have their own specific purpose. They do not enable a cooperative work among existing middleware systems.

The differences in the proposed business logic framework and the existing business logic models are as follows,

1. Integration
   – Collaboration of heterogeneous service systems
2. Independence and Flexibility
   – Work independently with a specific USN middleware
   – Gain flexibility when extending the platform infrastructure by enabling new collaborative services to work with the platform
3. Portability and Extendibility
   – Written in JAVA programming language
   – For mobile devices

4. Easiness
   – XML-based XLogic script language is used
   – Web services are used as a method of establishing connections with heterogeneous external systems
5. Real-time processing
   – A caching system is used

Shortens the time for parsing the scripts.

3. BUSINESS LOGIC FRAMEWORK

The business logic framework presented in this paper provides a scenario-based collaborative infrastructure that seamlessly integrates various application services in the RFID/USN environment. The business logic framework is composed of four agents: the Subscriber Agent, the Proxy Agent, the Instance Managing Agent, and the WAS Agent, with the additional Agent Manager. Upper-level application services collaborate with each other through the Agent Manager, which allows pre-registered scripts to be executed depending on the event data collected by the Subscriber Manager. The scripts are written in an XML infrastructure language entitled XLogic, which describes specific service scenarios and interacts with the business logic framework. Each agent may exchange event data under the supervision of the Agent Manager. The associated relationships of the agents are shown in Fig. 4.

Accordingly, events are processed in two modes within the business logic framework: a *trigger mode*, in which a received event will activate a corresponding XLogic script, and an *immediate mode*, in which an XLogic script is converted into a Java object and executed immediately after the script is registered in the business logic framework by the user.

Before we discuss the sub components of the business logic framework, we need to understand not only the internal events which are converted or generated within the platform, but also the common functions of the individual agents that are examined in the following section.

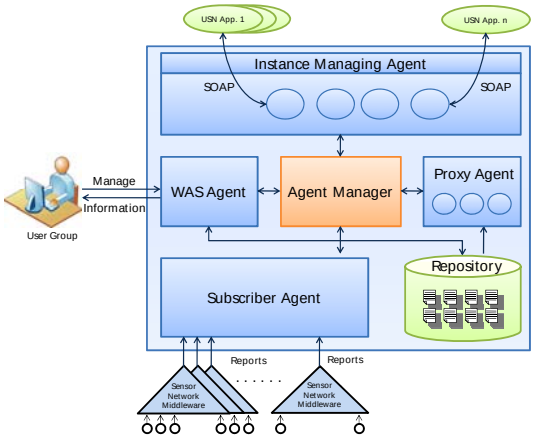


Fig. 4. The business logic framework structure.

3.1 The Internal Events Handling and the Common Functions

The business logic framework is written in Java to facilitate cross platform compatibility over heterogeneous systems. The interfaces are defined with SOAP. All of the execution units are driven by the event data within the business logic framework. Internal events are categorized by the following two events: "ReportsArrivedEvent" and "XLogicScriptExecuteRequestEvent" as shown in Table 2.

Table 2. Internal events.

Object	Description
ReportsArrivedEvent	Java object, converted from the event data to the XML reports by the Subscriber Agent
XLogicScriptExecute-RequestEvent	Java object holding its statistical information and the execution plan of XLogic script

The following table summarizes the common functions of the individual agents.

Table 3. Common functions of the agents.

Method	Function
getAgentName()	Return the name of an agent
Start()	Start an agent
Stop()	Stop an agent
getState()	Return the current status of an agent, one of the following four states, STARTED, STARTING, STOPPED, or STOPPING
processAgentEvent (AgentEvent event)	Process agent specific event

3.2 The Sub Components of the Business Logic Framework

This section starts by describing the Agent Manager and then explains the building blocks of the business logic framework, which is composed of the four agents of the Subscriber Agent, the Proxy Agent, the Instance Managing Agent, and the WAS Agent.

3.2.1 The agent manager

The Agent Manager is in charge of managing four agents and delivering every event received from each agent to the appropriate parties. Internal events simply pass through the Agent Manager. The Agent Manager propagates events to all of the agents registered to it, and the agents accept only events that are of interest for further processing and ignore other events.

3.2.2 The subscriber agent

Events originating from RFID/USN equipment are collected by querying the business logic framework for the following: a *snapshot query*, with which a USN middle-

ware requests immediate real-time sensor data, and a *continuous query* that is kept in an active state and is used to request sensor data continuously during a specified period.

The Subscriber Agent is connected to various types of USN middleware through multi-threaded socket connections on a pre-allocated port. It is in charge of forwarding the initially delivered sensor data to the Agent Manager after converting it into properly corresponding Java objects in a form known as a "ReportsArrivedEvent" event.

3.2.3 The proxy agent

The Proxy Agent is responsible for shortening the overall-event processing time. An XLogic script which has been executed is in memory as an executable Java object rather than the XML script itself saved in the repository, which reduces parsing time much more than if the same XLogic script is executed directly from the repository. This scheme makes it possible to manage system resources effectively throughout the entire lifetime of the XLogic script related to a specific service scenario.

The Proxy Agent usually extracts the unique identification information of the "ReportsArrivedEvent" event, which is forwarded by the Agent Manager. If the value matches one of the XLogic scripts resident in memory, the XLogic is converted into the "XLogicScriptExecuteRequestEvent" event and sent back to the Agent Manager. If it is not found in memory, XLogic in the XML form is queried and then parsed to be placed in memory as an executable Java object. Accordingly, the event is delivered to the Agent Manager in the manner described above.

3.2.4 The instance managing agent

The Instance Managing Agent not only executes the XLogic script but also maintains the statistics of the active XLogic, such as the success and failure ratio, the last completion time, and the current status. This information is presented online in real time through a web interface known as the Enterprise Manager (hereafter referred to as the EM) and helps the user establish an execution strategy based on the acquired statistical data.

3.2.5 The WAS agent

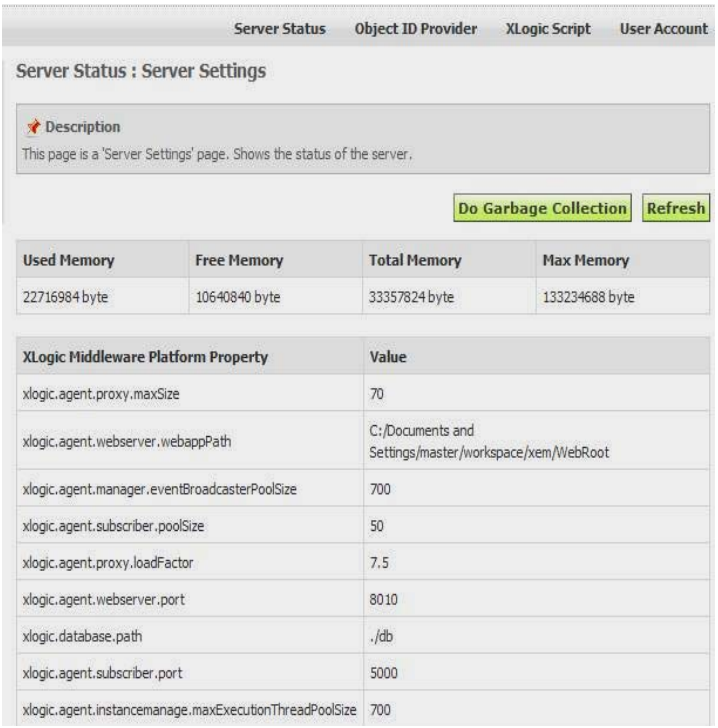
The WAS Agent exposes the useful functions to the outside by providing users with web services and acts as the web server of EM, as shown in Fig. 5.

3.3 XLogic Execution Modes

The users can register an XLogic script in either the trigger mode or the immediate mode of the business logic framework.

3.3.1 Trigger mode

An XLogic script, which runs in the trigger mode, is executed when an external event is collected by the business logic framework according to the flow of events depicted in the following figure.



Server Status | Object ID Provider | XLogic Script | User Account

Server Status : Server Settings

Description

This page is a 'Server Settings' page. Shows the status of the server.

Do Garbage Collection | Refresh

Used Memory	Free Memory	Total Memory	Max Memory
22716984 byte	10640840 byte	33357824 byte	133234688 byte

XLogic Middleware Platform Property	Value
xlogic.agent.proxy.maxSize	70
xlogic.agent.webserver.webappPath	C:/Documents and Settings/master/workspace/xem/WebRoot
xlogic.agent.manager.eventBroadcasterPoolSize	700
xlogic.agent.subscriber.poolSize	50
xlogic.agent.proxy.loadFactor	7.5
xlogic.agent.webserver.port	8010
xlogic.database.path	./db
xlogic.agent.subscriber.port	5000
xlogic.agent.instancemanage.maxExecutionThreadPoolSize	700

Fig. 5. An execution of the enterprise manager.

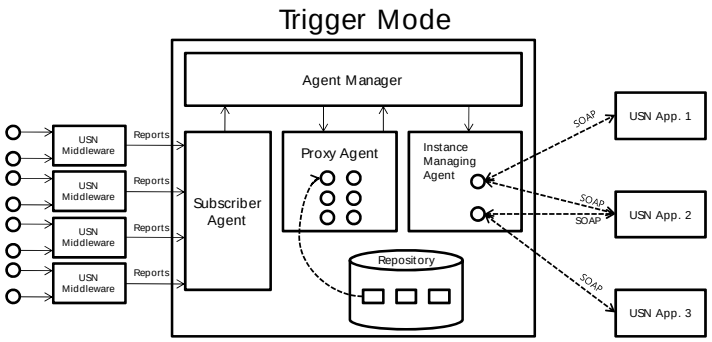


Fig. 6. Flow of events in trigger mode.

When an event is collected from the Subscriber Agent, the “ReportsArrivedEvent” event is sent to the Agent Manager. The Agent Manager forwards this to each agent registered to receive it. The Proxy Agent that is responsible for processing the “ReportsArrivedEvent” event adds statistical data and sends the ”XLogicScriptExecuteRequestEvent” event back to the Agent Manager, when it finds the related XLogic resident in memory. The Agent Manager sends the converted event to all of the agents again, and the Instance Managing Agent, capable of processing the “XLogicScriptExecuteRequestEvent” event, finally executes the XLogic.

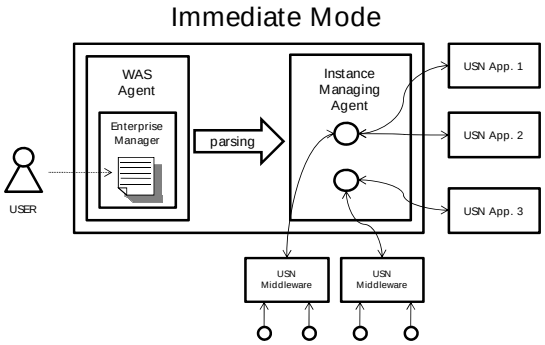


Fig. 7. Flow of events in immediate mode.

3.3.2 Immediate mode

This refers to the mode in which an XLogic script written by a user is executed immediately. If a user writes and executes an XLogic script in EM, the Instance Managing Agent transforms the XLogic script written in the XML form into a Java object and executes it according to the sequences shown in Fig. 7.

4. TEST SCENARIO

The following test scenario was devised to verify the functionalities of the business logic framework, especially for collaboration processes among heterogeneous application services.

1. Player “A” who is a member of the baseball team of “B” city, has arrived at the complex training center, “XX Corp.” and has been undergoing intense training.
2. “A” goes to the cafeteria in the center for breakfast.
3. Just as he enters the restaurant, he is recognized as a member of the “B” team because the Cafeteria Management System senses his RFID-enabled ID card.
4. After “A” places the food he wants on the tray, he passes the counter in which an RFID antenna is installed.

A. The food expense is calculated and transmitted to the Cafeteria Management System and is charged to the “B” team as part of the total food expenses at the end of the month.
B. The total calories in the food are calculated and sent to the Player Management System of the “B” team, which keeps track of the individual player’s health record.

The following XLogic script is presented to describe the above scenario.

```
<?xml version="1.0" encoding="UTF-8"?>
<xlogic:XLogicScript xmlns:xlogic="urn:spoon:xlogic:script:xsd:1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="urn:spoon:xlogic:script:xsd:1 http://spoon.smu.ac.kr/xlogic.xsd">
<!-- Bookmark 1 -->
<xlogic:set name="tags" select="//tag">${_REPORTS} </xlogic:set>
<!-- Bookmark 2 -->
<xlogic:invokeWebService service="PlayerInfoService"
     url="http://baseballteam.bcity.com/pm/baseball"
     name="find_player">
     <rawtags xmlns="bcity">
       <xlogic:iterator name="tag" source="tags">
         <tag>${tag}</tag>
       </xlogic:iterator>
     </rawtags>
  </xlogic:invokeWebService>
 <xlogic:set name="player" select="//id">${find_player}
  </xlogic:set>
<!-- Bookmark 3 -->
<xlogic:invokeWebService service="RestaurantPayService"
     url="http://restaurant.xxcorp.com/PayService"
     name="restaurant_pay_result">
       <foods xmlns="xxcorp">
         <xlogic:iterator name="tag" source="tags">
          <food>${tag}</food>
         </xlogic:iterator>
       </foods>
</xlogic:invokeWebService>
<!-- Bookmark 4 -->
<xlogic:invokeWebService service="CalorieService"
     url="http://restaurant.xxcorp.com/CalorieService"
     name="calorie_result">
       <foods xmlns="xxcorp">
            <xlogic:iterator name="tag" source="tags">
         <food>${tag}</food>
         </xlogic:iterator>
        </foods>
</xlogic:invokeWebService>
<xlogic:set name="calorie" select="//calorie">
    ${calorie_result}
</xlogic:set>
<!-- Bookmark 5 -->
<xlogic:invokeWebService service="PlayerManagementService"
    url="http://baseballteam.bcity.com/pm/baseball"
    name="manage_result">
  <updateFoodLog xmlns="bcity">
   <player>${player}</player>
   <calorie>${calorie}</calorie>
  </updateFoodLog>
```

```
</xlogic:invokeWebService>
</xlogic:XLogicScript>
```

The execution processes of the above XLogic script are as follows,

1. Tag data arrive at the Subscriber Agent of the business logic framework.
2. XLogic is executed according to the processing sequences of the trigger mode.
3. The captured tag data are set to a variable, called a tag (Bookmark 1).
4. From the tags, the player's tag information is set to variable, player (Bookmark 2).
5. The tag ID of the food is transmitted to the "RestaurantPayService" component of the Cafeteria Management System to accumulate the charges in the overall expenses (Bookmark 3).
6. The tag ID of the food is transmitted to the "CalorieService" component of the Cafeteria Management System, and all calories are calculated (Bookmark 4).
7. The calculated caloric value is sent to the "PlayerManagementService" component of the Player Management System to help the baseball team keep track of the nutrition history of the player (Bookmark 5).

In this test, the XLogic script above, designed to illustrate the interactions among heterogeneous application services, was successfully tested and shared among external services such as "RestaurantPayService," "CalorieService," and "PlayerManagementService" according to the anticipated execution sequences.

5. PERFORMANCE TESTS

To measure the performance of the business logic framework, the performance testbed shown in Fig. 8 was built. It is composed of reader-simulator supported by the BEA Weblogic RFID Edge Server Ver. 2.2, the business logic framework, and a performance monitor.

The first performance test measured the parsing time, which converts the XLogic scripts in XML form stored in the repository of the platform into a Java object. For this test, many XLogic scripts of variable lengths were presented as sample scripts. The measurement was accomplished by calculating the elapsed time between the beginning and

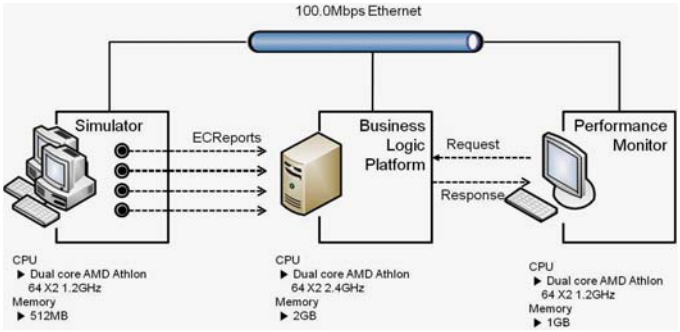


Fig. 8. Performance testbed.

end of the parsing of an XLogic script with the help of the system class of the java.lang package. To obtain the average elapsed parsing times, each XLogic was parsed 20 times. The following table shows the performance test results.

Table 4. Average parsing times per the number of lines of versatile XLogic scripts.

XLogic (Line)	Time (ms)
10	0.36
100	1.34
1000	3.92
1500	6.17
2000	8.83

The test results are helpful for the system administrator to estimate the execution time to parse the XLogic scripts. They can also be used as a measure to deploy collaborative services within the business logic framework infrastructure.

The second test examined the memory use ratio when the platform handles events on top of a JVM (Java Virtual Machine). In this test, the reader-simulator emulates the RFID reader equipment by generating ECReports of EPC (Electronic Product Code) data with GID-64 types, which are then forwarded to the business logic framework. The framework measures the allocated memory on the JVM, which has been configured to have a basic heap space of 32MB and a maximum heap space of 156MB, using the statistical-performance examining tool 'jvmstat' while running the XLogic scripts related to the ECReports. The number of executions of the XLogic script is displayed on the performance monitor, which is provided within the EM environment. The following script is activated for 10 seconds depending on the ECReports generated from 40 simulators.

```
<?xml version="1.0" encoding="UTF-8"?>
   <xlogic:XLogicScript
     xmlns:xlogic="urn:spoon:xlogic:script:xsd:1"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:spoon:xlogic:script:xsd:1
      http://spoon.smu.ac.kr/xlogic.xsd ">
        <xlogic:set name="tags" select="//tag">${_REPORTS}
        </xlogic:set>
        <xlogic:print>${"Test Script Start"}</xlogic:print>
        <xlogic:wait>${10000}</xlogic:wait>
        <xlogic:print>${"Test Script End"}</xlogic:print>
   </xlogic:XLogicScript>
```

The following figure shows the maximum number of instances of the XLogic scripts executed at a specific time, where the transmission period of the reports was set to 2 seconds. The pool size specifying the maximum number of instances of the Instance Managing Agent is configured, as the number of instances increases.

The width of the shaded box of Fig. 9 represents the execution time of an XLogic script and the height denoting the maximum number of instances which are still in the execution cycle on the JVM.

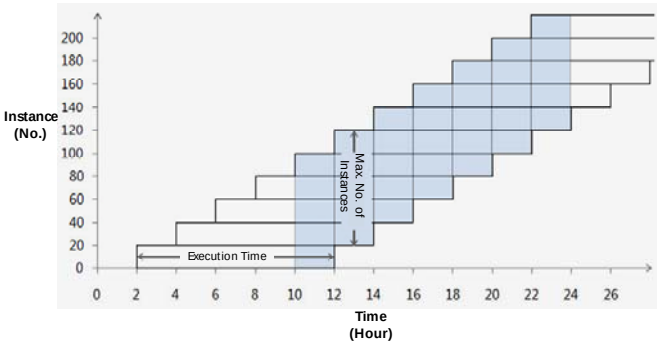


Fig. 9. Maximum number of instances of XLogic.

Table 5. The maximum number of instances and pool size depending on the report transmission period.

Report Transmission Period (Sec)	Maximum Number of Instances	Established Pool Size
3.0	160	200
2.0	200	250
1.0	400	450
0.9	440	500
0.8	500	550
0.7	560	600
0.6	640	700
0.5	800	900
0.4	1000	1050
0.3	1320	1350

The above table shows the maximum number of instances and pool size depending on the report transmission period.

To acquire the complete use ratio of the heap memory, the experiment lasted for 24 hours per report transmission period. The test results showing the use of the heap memory space depending on the report transmission period are as follows.

Table 6. The use of the heap memory space depending on report transmission period.

The Report Transmission Period (Sec)	Heap Memory Space (%)	Total Execution Time (Count)
3.0	29.562	1,151,695
2.0	31.787	1,727,893
1.0	34.579	3,455,931
0.9	35.824	3,839,817
0.8	41.341	4,319,952
0.7	53.239	4,937,131
0.6	62.199	5,759,932
0.5	65.840	6,911,935
0.4	70.543	8,639,538
0.3	79.578	11,518,936

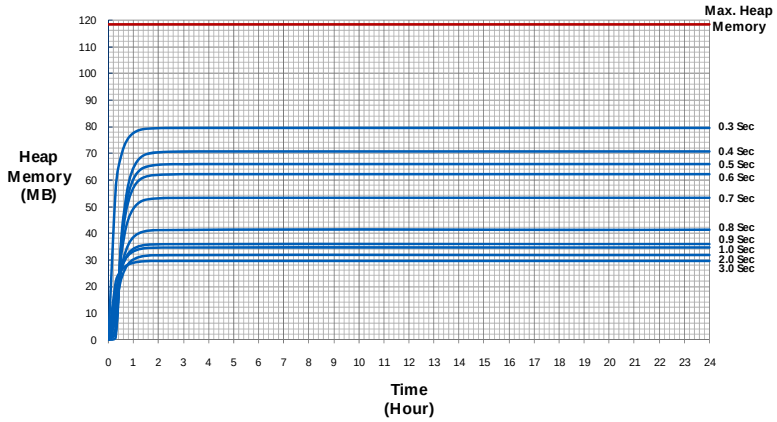


Fig. 10. Curves of the heap memory space depending on the report transmission period.

The initial and maximum heap memory space within the memory space of JVM can be specified as needed. The memory space may grow from the initial heap memory space, and if the space grows beyond the limits of the maximum memory space, memory deficiency occurs. Fig. 10 illustrates the curves of the heap memory space depending on the report transmission period. Whereas the memory use sharply increases in the beginning, the curve converges into a uniform line after a certain time in the figure.

This result shows that the business logic framework works reliably and consistently by properly configuring internal parameters depending on the number of transmitted reports.

6. CONCLUSIONS AND FUTURE WORKS

To realize a highly networked telecommunication environment, numerous researchers have focused on the core infrastructures running on the RFID/USN-enabled ubiquitous environment. As a result, various middleware platforms that integrate many business applications have been actively proposed by many software vendors and research institutions. Moreover, thousands of application services have begun to operate on top of these middleware platforms. Therefore, it is very important to develop a collaboration framework as well as a specific matching middleware platform. The business logic framework proposed in this paper enables a cooperative work among existing middleware systems and enables a variety of RFID/USN-enabled services to be combined within the flexible framework to provide composite and refined services.

In this paper, our research team proposes an effective means of putting variable heterogeneous systems together within an infrastructure environment and making them organically flow through the business logic framework without disturbing any autonomy belonging to the individual application services. The services are triggered by pre-registered scenarios easily customized by users using the XLogic script language, as they expect them to work in the RFID/USN environment. This research shows that the platform successfully demonstrates its anticipated capabilities, such as seamless integration with external services. Moreover, its performance is stabilized when it serves as a collabora-

tion infrastructure framework.

However, the requirement to learn XML to write service scenarios may cause be inconvenient in some cases. For this reason, the research team continues to extend the platform by providing a visualized scenario production tool for more convenient user accessibility. Furthermore, the research will focus on enhancing the performance of the business logic framework and seeking better collaboration mechanisms among heterogeneous service systems.

REFERENCES

1. W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *IEEE Network Magazine Special Issue*, Vol. 18, 2004, pp. 6-14.
2. G. J. Kim, S. J. Kim, N. S. Kim, and C. S. Pyo, "USN service and market trend," *Journal of Korea Information Science Society*, Vol. 25, 2007, pp. 7-18.
3. M. S. Kim, Y. J. Lee, and J. H. Park, "Trend of USN middleware technology," *Journal of Electronics and Telecommunications Research Institute*, Vol. 22, 2007, pp. 67-79.
4. Y. B. Kim, C. S. Kim, and J. W. Lee, "A middleware platform based on multi-agents for u-healthcare services with sensor networks," in *Proceedings of ACM Symposium on Applied Computing*, 2008, pp. 683-692.
5. S. Li, S. H. Son, and J. A. Stankovic, "Event detection services using data service middleware in distributed sensor networks," in *Proceedings of the 2nd International Conference on Information Processing in Sensor Networks*, Vol. 26, 2003, pp. 502-517.
6. T. Liu and M. Martonosi, "Impala: A middleware system for managing autonomic, parallel sensor systems," in *Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2003, pp. 107-118.
7. S. R. Madden, M. J. Franklin, and J. M. Hellerstein, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, Vol. 30, 2005, pp. 122-173.
8. J. H. Paik, H. S. Kim, Y. H. Kim, and S. I. Han, "New technology trends on indexing for moving objects," *Journal of KISS*, Vol. 25, 2007, pp. 40-46.
9. K. Park, Y. Kim, J. Lee, and J. Chang, "Integrated design of event stream service system architecture," in *Proceedings of International Conference on E-Business*, 2007, pp. 51-56.
10. K. Park, Y. Kim, J. Chang, D. Rhee, and J. Lee, "The prototype of the massive events streams service architecture and its application," in *Proceedings of the 9th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 2008, pp. 846-851.
11. Y. Yao and J. E. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM SIGMD Record*, Vol. 31, 2002, pp. 9-18.
12. Oracle, Gartner, JavaOne Conference in San Francisco, 2006.
13. An Oracle White Paper "The instantly responsive enterprise: Integrating business process management and complex event processing," 2008, http://www.oracle.com/

technologies/bpm/docs/instantly-responsive-enterprise-whitepaper.pdf.
14. BEA, "BEA event driven SOA solution," http://kr.bea.com/BEA_SolutionGuide_2.pdf.
15. Global Headquarters 3303 Hillview Avenue Palo Alto, CA 94304 "Event-driven SOA: A better way to SOA," http://www.tibco.com/resources/solutions/soa/eventdriven_soa_wp.pdf.

Seok-Kyoo Kim received his M.S. and B.S. degrees in Computer Science from Seoul National University, Seoul, Korea, in 1994 and 1992, and the Ph.D. degree in Computer Science and Engineering from Seoul National University, in 2010. He has been worked for NC Soft from 1999 to 2003. He is a senior researcher of Korea Institute of Science and Technology Information, since 2011. His research interests include middleware, digital storytelling, computer game, HCI and augmented reality.

Juno Chang received his Ph.D., M.S. and B.S. degrees in Computer Science from Seoul National University, Seoul, Korea, in 1990, 1992 and 1998, respectively. He was consulting director in i2 Technologies from 1998 to 2003. He is a Professor of Sangmyung University, Seoul, Korea, since 2003. His areas of research include business S/W, middleware, and digital storytelling.

Sang-Yong Han received the B.S. degree in Applied Mathematics from Seoul National University, Seoul, Korea, in 1972, the M.S. degree in Computer Science from Seoul National University, in 1977, and the Ph.D. degree in Computer Science from University of Texas, Austin, TX, in 1983. He is a Professor of the School of Computer Science and Engineering of the Seoul National University, Seoul, Korea, since 1984. His areas of research include middleware, parallel processing and dataflow computing.