**TELKOMNIKA Indonesian Journal of Electrical Engineering**
Vol. 12, No. 10, October 2014, pp. 7395 ~ 7403
DOI: 10.11591/telkomnika.v12i8.5352

# Networked Automatic Test System Based on Message-Oriented Middleware

**Xiong Fengguang*, Han Xie**
North University of China Computer and Control Engineering, Taiyuan 030051, China
*Corresponding author, e-mail: xfgncit98@sina.com

***Abstract***

*Aim at the current status of automatic test system that focuses only on a single or the same type of weapons' performance testing, a networked automatic test system based on message-oriented middleware is proposed in this paper, and inner architecture and process of pus/sub in message-oriented middleware are described. Basing on the message-oriented middleware, the networked automatic test system can shield the complexity and diversity of the bottom test system and test equipment, and can realize cross-platform communication of various test data about weapons. Experiments show that the networked automatic test system based on message-oriented middleware is an integrated test platform which can provide comprehensive performance test for coordinated engagement on a variety of weapons.*

***Keywords:*** *networked automatic test system, test data, message-oriented middleware, message publishing, message receiving, message subscription*

## 1. Introduction

Rapid development and application of key information technology promotes a new revolution in military affairs and development of information warfare, and promotes development and revolution of modern military theory and operational model, such as a network-centric warfare. The prerequisite of winning in the future wars depends on whether cooperative engagement between weapon platforms can be implemented. Therefore, building an integrated test platform for comprehensive performance test of a variety of weapons' cooperative engagement is a crucial task.

General speaking, the integrated measurement and device which are as core of computer and can automatically perform certain test task under the control of procedure are called automatic test system [1, 2] (ATS). In order to improve the modernization level of airborne equipment maintenance, the research of automatic test system has become the focus of development in the world military equipment [3]. Current ATS has been widely applied to a variety of weapon and equipment tests and has become necessary assurance of weapon' reliable operation. However, the current ATS is different from the military, and is lack of interoperability between different systems. So, ATS can't adapt to the modern jointness's demand on multi-weapon systems, multi-level maintenance.

This paper aims to build a networked ATS based on message-oriented middleware [4, 5] (MOM). The message-based middleware is between the bottom equipment and application systems, and shields complexity and diversity of the bottom test system, and achieves data's effective transmission and wide sharing on the internet through MOM's strong function of sending and receiving data, and enhances interaction of networked ATS and reliability of weapons' coordination test.

## 2. Architecture of Automatic Test System

In general, ATS is made of hardware and software. Hardware is made of host computer, general board, test instrument and measured object. Board and test instrument are responsible to control equipment or for signal acquisition, and can be based on various data bus, such as PCI, PXI, GPIB. The measured object refers to any test equipment, systems or subsystems, production lines, etc. In the process of putting up test system, user usually pays

*Received December 13, 2013; Revised June 24, 2014; Accepted July 20, 2014*

attention to the operation of test board and instrument. Software is made of I/O interface, drivers of board and instrument, executable application. I/O interface provides the interface of the bottom hardware driver and is responsible for communications between the host computer and the physical instrument. Instrument driver is a software assembly between the upper software and a specific instrument, and is responsible for transferring instruction between application and instrument, and is a middle communication layer between I/O interface and application. Executable application promotes user-friendly interaction interface to execute test operation by instrument driver, and promotes data analysis, data processing, and data storage. I/O interface, board and instrument driver, executable application constitutes the entire software architecture of automatic test system. The architecture is shown as Figure 1.

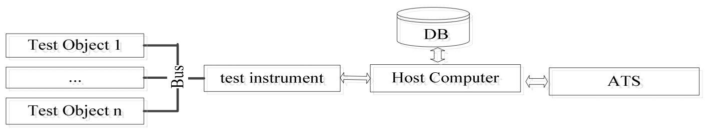

Figure 1. Architecture of Automatic Test System

## 3. Architecture of Networked Automatic Test System Based on Message-oriented Middleware

Networked automatic test system [6] (NATS) is a kind of automatic test system built on network. Through message-oriented middleware and network, NATS based on message-oriented middleware can build a networked test system. Each node in the network is a host computer which obtains test information through the test bus connecting test equipment. Each host computer can receive additional information sent by the other host computer through message-oriented middleware, and also publishes information to other host computer that subscribes the information. Its architecture is shown as Figure 2.

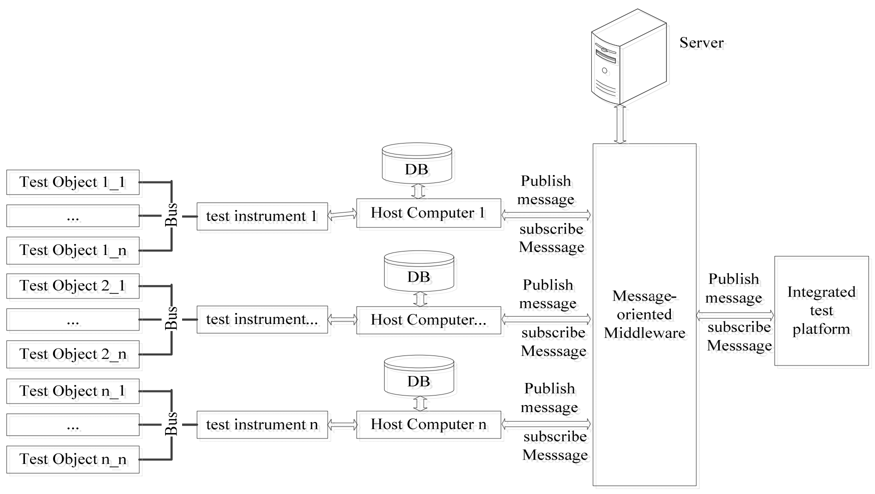

Figure 2. Architecture of Networked Automatic Test System Based on Message-oriented Middleware

Host computer, connected to the specific test instrument in order to obtain test information of various test instruments, is a node in the NATS. After obtaining test information, host computer also need to analyze, show, store and send those information. Other, each host computer can subscribe test information of other host computer. That is to say, different host computer can realize interaction with real information and can complete coordination test of weapon. Message-oriented middleware is a bridge for test information exchange, and is responsible for receiving data published by host computer and distributing information to subscribers which include host computer s or integrated test platform. Integrated test platform is a set of application system in the server. Through message-oriented middleware, integrated test platform can receive test information of each host computer in the network according to demand, and also can send command to each host computer to make it test specific information for integrated test platform. Through amount of test data [7] from different host computer, integrated test platform can realize coordination test of weapons which includes analysis of test data, simulation of test process, real-time display and so on.

## 4. Design and Implementation of Automatic Test System Based on Message-oriented Middleware

Messaging middleware is responsible for sending and receiving messages on different application systems, and can realize platform-independent data exchange. Communicating parties send and receive message in synchronous or asynchronous mode, so actual physical access is unnecessary. Therefore, the message-oriented middleware is adapted to couple loose host computer up in order to interactive communication and coordinated test.

### 4.1. Design and Implementation of Message-oriented Middleware's Architecture

The key technology of message-oriented middleware is message transfer, and the model of message transfer includes point to point and publish-subscribe.

a) Point-to-point model

In this model, the producer of message is called sender, and the consumer of message is called receiver. The sender sends a message to a queue and is stored in the queue, and the receiver obtains message from the queue. The disadvantages of this model is that the message sent by sender can only be received by only a receiver, and it severely limits the use range and makes information sharing poor.

b) Pub/sub model [8]

In this model, consumer of message is called as publisher and consumer of message is called as describer. The difference with point-to-point is that a message published to a subject can be received by more describers. Pub/sub model is a model based on push, and message can automatically broadcast to consumers. This model can ensure that message is shared widely and is able to adapt for transfer of test data between different host computers. The architecture of message-oriented middleware based on pub/sub model is shown as Figure 3.

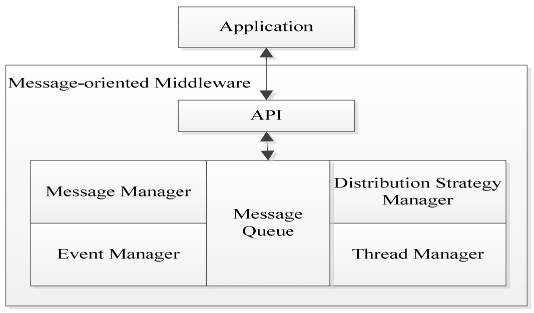

Figure 3. Architecture of Message-oriented Middleware

c) Message queue

Message queue is a communication model of asynchronous transfer mode between different applications, and includes two conceptions: message and queue.

Message is a carrier of test data, and mainly consists of message head and message body. The architecture of message is shown as Figure 4. Message head is metadata of message, and is necessary information when message delivers, and includes: ID, flag, type, publishing date, expire date, persistence, priority, publisher, receiver, length, and so on. The code is shown as following:

```
public class Message {
public long ID;                         //Message's ID
public int flag;                                //Message's Flag
public int type;                        //Message's type
public Date publishing_date;            //Message's publishing date
        public Date expire_date;                //Message's expire data
public boolean persistence;             //Message's state of persistence, the value is
true/false
public byte priority;                           //Message's priority
public SocketChannel publisher;//Publisher of Message
public SocketChannel receiver;  //Subscriber of Message
public int length;                              //Message's length
public byte[] data;                             //Message's content
}
```

Message can be divided into event message and normal message, and we can distinguish them by the value of "flag", in the message head, whose value is 0 that denotes the message is event message and is 1 that denotes the message is normal message. Event message is divided into register and unregister of application, subscription and unsubscription of distribution strategy, and we can distinguish them by the value of "type", in the message head, whose value is 0 that denotes register of application, and is 1 that denotes unregister of application, and is 2 denotes subscription of distribution strategy, and is 3 that denotes unsubscription of distribution strategy. Normal message encapsulates test data of information exchange, and has 256 kinds of types, and we can also distinguish them by the value of "type" in the message head. In general, we can distinguish any kinds of message according to "flag" and "type" in the message head. In addition, "data" is message body and its content is test data which can be text information, voice, image and other binary information.

Queue is also a public storage space and it exists in memory or physical file, but generally it exists in memory in order to ensure the speed of message transfer. Queue can be divided into sender queue, event queue and waiting queue. Sender queue is used to storing normal message. On the one hand, message-oriented middleware pushes the normal message received from the publisher of normal message into the sender queue, but on the other hand, message-oriented middleware also obtains message from the sender queue and sends it to the subscribers in order to realize message transfer. Event queue is used to storing event message. On the one hand, message-oriented middleware pushes the event message received from the publisher of event message into the event queue, but on the other hand, message-oriented middleware also obtains event message and make it handled by event manager. Waiting queue is used to storing normal message unsuccessfully sent to subscribers. The main reason that normal message can't be sent to subscribers is that communication between host computers disconnects. So, once connection resumes, normal message in waiting queue can be sent to the subscribers in order to ensure the reliability of message transfer.

d) Queue manager

Queue manager is responsible for managing all event queues which includes pushing the message into queue, popping message from queue. In addition, in order to ensure the efficiency of pushing and popping message, multi-thread is used to managing queue; the method of pushing and popping message is synchronized in order to ensure pushing and popping message in order at the same time.

e) Distribution strategy manager

How to distribute normal message to subscribers is determined by distribution strategy. So, according to the difference of requirement for message, different NATS will subscribe different distribution strategy which forms a table of distribution strategy named "T_DistributionStrategy". Class of distribution strategy is shown as following.

```
public class DistributionStrategy {
        SocketChannel socket;
        int[] types;
}
```

In this class, "socket" object is a socket connecting NATS with message-oriented middleware, and is used to describing IP and port of a subscriber; "types" is an array in which all subscribed message type of a subscriber are included. The function of distribution strategy manager is to insert a distribution strategy into "T_DistributionStrategy", and update date in "T_DistributionStrategy" through a distribution strategy, and delete data from "T_DistributionStrategy" through a distribution strategy.

f) Event manager

Event manager is mainly used to processing the application's event requests, including event receiver, security detector, event handler, and response notifier. The process flow is generally divided into the following steps:

Step 1: event receiver obtains an event message from event queue, and then security detector checks whether the event message meet the system's security; if the detection does not pass, response notifier notifies the result to the application, and if the detection passes, the process forwards step 2;

Step 2: Parsing the value of "type" from the event message; if the value is 0 that denotes the event is register and process forwards step 3; if the value is 1 that denotes the event is unregister and process forwards step 4; if the value is 2 that denotes the event is subscription of distribution strategy and process forwards step 5; if the value is 3 that denotes the event is unsubscription of distribution strategy and process forwards step 6;

Step 3: Generating an object of distribution strategy that contains a "socket" object, and a "types" variable whose value is null; and then, calling distribution strategy manager to add the object into "T_DistributionStrategy", and process forwards step 7;

Step 4: Comparing one by one "socket" object in distribution strategy from "T_DistributionStrategy" with "socket" object in the event message; if the comparison is successful, the matched distribution strategy from "T_DistributionStrategy" will be deleted from the table according to calling delete function of distribution strategy manager, and process forwards step 7);

Step 5: Comparing one by one "socket" object in distribution strategy from "T_DistributionStrategy" with "socket" object in the event message; if the comparison is successful, the matched distribution strategy from "T_DistributionStrategy" will be updated by the event message object according to calling update function of distribution strategy manager to add the "data" in the event message to the "type" variable in the matched distribution strategy object, and process forwards step 7);

Step 6: Comparing one by one "socket" object in distribution strategy from "T_DistributionStrategy" with "socket" object in the event message; if the comparison is successful, the matched distribution strategy from "T_DistributionStrategy" will be updated by the event message object according to calling update function of distribution strategy manager to delete some values of "type" variable in the matched distribution strategy object according to the "data" in event message, and process forwards step 7);

Step 7: Response notifier notifies the result to NATS.

g) Thread manager

In message-oriented middleware, in order to ensure the efficiency of message processing, the enqueue (pushing message into queue) and dequeue (popping message from queue) of event message and normal message are operating at the respective thread. Among them, the logic process of enqueue and dequeue of normal message is not the same and execution environment is not the same, so for the sending queue in per unit time, the amount of enqueue and dequeue is certainly inconsistent. We use to the ratio of the number of enqueue's message and dequeue's message to describe the throughput efficiency of sending queue, and it is shown as formula 1.

$$\text{Throughput} = \text{number of normal message pushed into query per second} / \text{number of normal message popping from query per second} \quad (1)$$

Meanwhile, considering the number of normal message is enormous, if throughput efficiency continues low, an amount of normal message will be piled up in the sending queue over time. On the one hand, it will affect the efficiency of the implementation of message-oriented middleware; on the other hand it can also make application receive a message in delay. Thus, by adding thread management into message-oriented middleware and increasing or reducing the number of dequeue' thread according to throughput in order to balance the efficiency of enqueue and dequeue of normal message. The number of dequeue's thread is shown in formula 2.

$$\text{Thread number used to popping message from query} = \begin{cases} 1 & (\text{length of query} < 2000) \\ 2 & (\text{length of query} \geq 2000, \text{ and } 2 \leq \text{throughput} < 3) \\ 3 & (\text{length of query} \geq 3000, \text{and } 3 \leq \text{throughput} < 4) \\ 4 & (\text{length of query} \geq 4000, \text{and } 4 \leq \text{throughput} < 5) \\ 5 & (\text{length of query}5000, \text{and throughput} \geq 5) \end{cases} \quad (2)$$

### 4.2. Pub/sub of Message-oriented Middleware in NATS

a) Constituent element of message's pub/sub

Pub/sub of message in NATS consists of publisher, subscriber, message center and table of distribution strategy, so we can describe the pub/sub of message with a four-tuple (P, S, C, T). P is a set of the message's publisher {p1, p2, ..., pn}, and S is a set of message's subscriber {s1, s2, ..., sn}. P may intersect with S, so that is to say that the message's publisher and message's subscriber are not absolute, and they can be a unity (both a publisher and a describer). C is the message's center, and consists of message queue, queue manager, event manager, thread manager, distribution strategy manager, and is responsible for contact message's publisher and subscriber. Not only message-oriented middleware receives message published by message's publisher, but also sends message to message's subscriber. When C is equal to 1, all of the message's publishing and subscription use a message center, and it is the simplest message's pub/sub system. Usually, within NATS there are multiple message centers, so C is greater than 1. T is a table of distribution strategy which ensures correct distribution of message between publishers with subscribers.

b) Process of message's pub/sub

Step 1: NATS starts the function of message' publishing, and then generates a message: "msg", and then sets the value of "flag" in the message to 1 (It sets the message as a normal message in fact), and then sets the value of other variable in the message, at last, set test data to "msg" in the message;

Step 2: NATS sends the "msg" to message center;

Step 3: After message's center receiving the message, according that the flag's value is one, the message can be determined as a normal message, and then parsing the type of message judges whether a "types" in some distribution strategy contains the type of message; if a "types" in some distribution strategy contains the type of message, then there is ATS subscribing the message, and then according to queue manager ,the message is pushed into sending queue; if not a "types" in some distribution strategy contains the type of message, so there is no subscriber to subscribe the message and there is unnecessary to push the message into sending queue;

Step 4: Message's center parses the value of "persistence" in message; if the value is one then the message is stored in the database, otherwise the message does not store in the database;

Step 5: Message's center responses to ATS that the message is received successfully.

c) Main process of message' subscription

Step 1: ATS starts the function of message's subscription;

Step 2: Popping a message from the sending queue, and setting the message as "msg", and parsing the "type" in the "msg", and finally getting a distribution strategy from the "T_DistributionStrategy" and then naming it as "ds ";

Step 3: If the "types" in "ds" contains the type of message, then according to "socket" in "ds", sending the message to subscribing ATS; if sending the message unsuccessfully, then assigning the socket to the "receiver" in message and pushing it to waiting queue;

Step 4: If existing unvisited distribution strategy in "T_DistributionStrategy", then getting a next distribution strategy and naming it as "ds", and jumping step 3; if all the distribution strategies are visited, then jumping step 5;

Step 5: If there is message in sending queue, then jumping step 2; if there is not message in sending queue, then waiting until receiving message and jumping step 2.

d) Main process of subscribing and distributing

Step 1: ATS starts the function of subscription and distribution;

Step 2: ATS generates a message named as "msg", and sets the flag of "msg" to zero, and sets the type of "msg" to two, and convert the content of message as byte array and assigning it to data of "msg";

Step 3: ATS sends "msg" to message's center;

Step 4: After message's center receiving the message, according that the flag's value is zero, the message can be determined as an event message, and then calling event manager to complete subscription of distribution strategy.

e) Main process of unsubscribing distribution strategy

Step 1: ATS starts the function of unsubscribing distribution strategy;

Step 2: ATS generates a message named as "msg", and sets the flag of "msg" to zero, and sets the type of "msg" to three, and convert the content of message as byte array and assigning it to data of "msg";

Step 3: ATS sends "msg" to message's center;

Step 4: After message's center receiving the message, according that the flag's value is zero, the message can be determined as an event message, and then calling event manager to complete unsubscription of distribution strategy.

## 5. Application of ATS Based on Message-oriented Middleware

For example, in ATS of coordinated engagement of air-ground weapon which is show as Figure 6, first aerial reconnaissance investigates and locates enemy's targets in air, sea and land; and then enemy's information is published via message-oriented middleware, and ground control center subscribes the enemy's information while ground control center also subscribes longitude, altitude, height, speed, azimuth and other information which are published by unmanned aerial vehicles and fighter, as well as location, transmit status which are published by precision-guided missiles; after analyzing the information published by aerial reconnaissance, unmanned aerial vehicles, fighter and precision-guided missiles, ground control center publishes information of damaged target to message-oriented middleware, and based on the distribution strategy, message-oriented middleware distributes the information to precision-guided missile, fighter and unmanned aerial vehicles; finally, unmanned aerial vehicles, fighter and precision-guided missiles attack the target and publish the damaged information to message-oriented middleware, and evaluation control center subscribes the damage information, and conducts a comprehensive analysis.

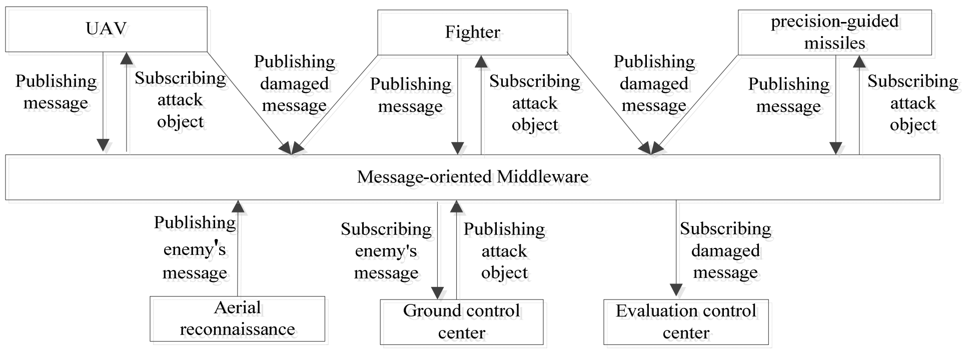
UAV
Fighter
precision-guided missiles
Publishing message
Subscribing attack object
Publishing damaged message
Publishing message
Subscribing attack object
Publishing damaged message
Publishing message
Subscribing attack object
Message-oriented Middleware
Publishing enemy's message
Subscribing enemy's message
Publishing attack object
Subscribing damaged message
Aerial reconnaissance
Ground control center
Evaluation control center

Figure 6. ATS of Coordinated Engagement of Air-ground Weapon

## 6. Test and Analysis

### 6.1. Test Target

Message-oriented middleware mainly provides a common information communication platform for ATS, so test mainly considers two aspects: first, the time that message-oriented middleware distributes messages to subscribers whether can meet the requirements; second, the middleware system can whether distribute accurately messages to the specified subscribers.

### 6.2. Test Environment and Test Data

Software Environment: Windows XP operating system and ATS of coordinated engagement of air-ground weapon.

Hardware Environment: There are 7 computers, and a computer is used as a message-oriented middleware server, and other six computers are used to simulating the host computer of unmanned aerial vehicles, fighter, precision-guided missile, aerial reconnaissance, ground control center and evaluation control center. Those computers' configurations are Intel Core2 Quard CPU 2.40 GHZ and RAM 2G.

The host computer of aerial reconnaissance publishes information of enemy, the host computer of unmanned aerial vehicles, aircraft and precision-guided missiles publish their information, when the length of message in sending queue is equal to 2000, message-oriented middleware starts to distribute message, while the host computer of aerial reconnaissance, unmanned aerial vehicles and precision-guided missiles stop publishing information, and distribution time will be record in Table 1. In the same way, distribution time of distributing 4000, 5000 and 10000 messages will be record in Table 1.

Table 1. Distribution Time of Message-oriented Middleware

| | *2000 Message* | *4000 Message* | *5000 Message* | *10000 Message* |
|---|---|---|---|---|
| *total distribution time（Millisecond）* | *120159* | *232877* | *311362* | *654186* |
| *Distribution time per record（Millisecond）* | *60* | *58* | *62* | *65* |

The host computer of unmanned aerial vehicles, aircraft and precision-guided missiles publish damaged information 1000 times. The statement of distribution message is shown in Table 2.

Table 2. Statement of Distributing Message

| *Sender* | *host computer of unmanned aerial vehicles,* | *Host computer of aircraft* | *Host computer of precision-guided missiles* |
|---|---|---|---|
| *Receiving times* | 1000 | 1000 | 1000 |

### 6.3. Result Analysis

By comparing these two sets of above test, we can draw the conclusions:

(1) The basic distribution efficiency of message-oriented middleware is about 60s/record, and distributing 4000 message wastes at the least, which shows that the number of threads of distributing message is optimal.

(2) The Accuracy rate of message-oriented middleware distributing message is 100%, and this indicates that the distribution accuracy of the message-oriented middleware is reliable.

## 7. Conclusion

Aim at the current status of the ATS that focuses on only on a single or the same type of weapons' performance testing, this paper introduces message-oriented middleware into ATS in order to not only shield complexity of the bottom test equipment, but also implement test data transfer between different test equipment through the message's publish and subscription. According to test on message-oriented middleware about publishing and subscribing message we can draw the conclusion that the message-oriented middleware can create an integrated test platform for a variety of weapons and implement comprehensive performance test for weapon's coordinated engagement.

## References

[1] Yingliang Feng, ShujuanWang, Bin Feng, RuiWang, Yuan He, Tong Zhang. Development of an auto test system for humidity sensors. *Sensors and Actuators A: Physical*. 2009; 152(1): 104-109.
[2] Lai Qing, Hua Jian-Wei, Lü Yun, Chen Yue-Fei, Xu Jian. Research on general relay protection auto-test system software. *Power System Protection and Control*. Dianli Xitong Baohu yu Kongzhi. 2010; 38(3): 90-94.
[3] Yuan Qingfeng, Lu Hui, Shen Shituan. Auto test system resource description method based on XML. *Journal of Beijing University of Aeronautics and Astronautics*. Beijing Hangkong Hangtian Daxue Xuebao. 2010; 36(1): 114-117.
[4] Zhai Lifang, Li Chunyuan, Sun Liping. Research on the message-oriented middleware for wireless sensor networks. *Journal of Computers*. 2011; 6(5): 1040-1046.
[5] Miao Chun-Yu, Shi Mei-Lin, Jiang Jin-Lei. MOM-S: Message-oriented middleware based on Web service. *Journal on Communications*. Tongxin Xuebao. 2006; 27(11): 96-100.
[6] Jia Zhiyan, Han Xie. Network automatic test system based on database middleware. Lecture Notes in Electrical Engineering. 2013; 218(3): 603-609.
[7] Jian Ni, Yunlong Zhang. Study of test data generation method based on evolutionary algorithm. *Telkomnika*. 2014; 12(1): 818-822.
[8] Carrilho Sergio, Esaki Hiroshi. A pub/sub message distribution architecture for disruption tolerant networks. *IEICE Transactions on Information and Systems*. 2009; E92-D(10): 1888-1896.