Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A framework for Internet service evolution based on active object

HU Hua[†1], ZHANG Yang[2]

([1]*Institute of Computer & Information Engineering, Zhejiang Gongshang University, Hangzhou 310035, China*)

([2]*School of Computer Science & Engineering, South China University of Technology, Guangzhou 510640, China*)

[†]E-mail: huhua@mail.zjgsu.edu.cn

Received Jan. 19, 2006; revision accepted June 16, 2006

Abstract: The wide use of Internet Service in distributed computing and e-business has made the evolution of Internet Service to be one of the most prevalent research fields in software development domain. Traditional methods for software development cannot adapt to the challenge of Internet Service oriented software development. In this paper, we propose a new paradigm for the evolution of Internet Service with active objects from the characteristics of Internet Service and principles of active objects. The paradigm uses an automatic monitoring mechanism of active object to detect and process evolution requirement in system based on Internet Service.

Key words: Framework, Internet Service, Evolution, Active object
doi:10.1631/jzus.2006.A1662 **Document code:** A **CLC number:** TP311.52

INTRODUCTION

The rapid development of Internet and E-business technology has obviously stimulated the wide use of online Internet Services whose increasing number has stimulated much research effort toward online Internet Services software development, and also attracted much attention on the evolution of Internet Service oriented systems. Traditional software developing paradigms, such as waterfall paradigm or prototype paradigm, have problems with adapting to the challenge of online software evolution. This generates a new requirement to develop new technologies for evolution of online Internet Service. Gadke and Graef (2000) proposed a Web Engineering paradigm for software development, where a model of Web composition is used to fulfill the development and evolution of Web software. Ingham *et al*.(1997) used the Web Engineering paradigm to develop Web Services. Chen *et al*.(2002) and Kicman and Fox (2004) proposed a feedback mechanism to help the evolution of Internet Service. Kirda *et al*.(2001) used engineering management and distributed methods to support application of Web services. The main focus of Web Engineering was the development of Web application based on software components, which made Web Engineering research stay in the fields of software component reusing and rapid composition of Web application. Chen's feedback mechanism depended mainly on human beings and lacked a standard and automatic mechanism. While the current Internet Service software researches cannot adapt to the requirement from the huge number of Internet Services, this paper, based on the characteristics of Internet Services and principles of active objects, proposes a new paradigm for the evolution of Internet Service with active objects. This paradigm uses an automatic monitoring mechanism for active objects to detect and process evolution requirements in an Internet Service system.

This paper is organized as follows. Section 2 introduces the basic theory of active objects. In Section 3, a new Internet Service Evolution (ISE) paradigm is proposed. Section 4 details the key

[*] Project (No. 2005C21025) supported by the Science and Technology Department of Zhejiang Province, China

components of the new paradigm. Section 5 gives a simulation example and Section 6 concludes the paper.

INTERNET SERVICE ORIENTED ACTIVE OBJECT

Event Condition Action (ECA) is a general rule model for researchers of active database to describe behaviors of active databases. In ECA model, "E" represents events that happen in the environment of the active database; "C" stands for the conditions for the active database to respond to the occurring event; "A" represents active database actions triggered by the events when conditions are true. By separating and abstracting the strategy and policy of a system from system data to ECA rules, an active database based on ECA model can realize the separation of management knowledge from normal application data. However, when we try to incorporate ECA into ISE, we found that ECA model cannot be directly applied to active objects for ISE, because an active object for ISE is a distributed active entity for system control and management. The attributes of a distributed object make the behaviors of active objects for ISE closely related with the changes of time, space and system internal states. It is the time and space problems of active objects that resulted in the many limitations ECA model has in describing behaviors of active objects in the Internet. These problems inspired us to extend the ECA model to the EECA (Extended ECA) model by adding time and space description ability to ECA. EECA model can be used to describe active object on the Internet due to its ability to describe time and space domain.

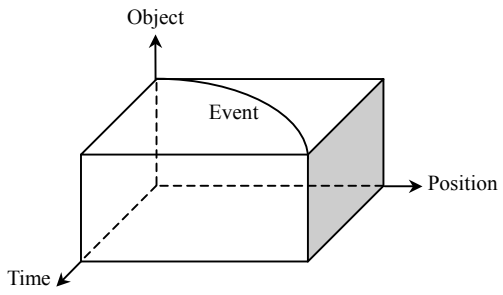

Fig.1 Time and space specialty of the active object

In the following text, we use $\alpha=\{e_0, e_1, e_2, \ldots, e_n\}$ to represent the set of all the events of a system, $\beta=\{\chi_0, \chi_1, \chi_2, \ldots\}$ to represent the attribute space of all the objects in the system, $\gamma=\{\xi_0, \xi_1, \xi_2, \ldots\}$ to represent the attribute space of the environment in which the objects are running.

Definition 1 An event for active object is a triple-tuple $\{e, t, p\}$, where $e\in\alpha$, t are time stamps with structure of yyyymmddhhmmssms, and p is a URL location.

Definition 2 The internal state of an active object is a five-tuple $\{T, \chi_s, \chi_d, t, p\}$, where T is a global unique transaction identifier, and χ_s, $\chi_d\in\beta$ are static and dynamic attributes of an active object, respectively. The meanings of t and p are the same as those defined in Definition 1.

Definition 3 The external state of an active object is a five-tuple $\{T, \xi_s, \xi_d, t, p\}$, where ξ_s, $\xi_d\in\gamma$ are static and dynamic attributes of an active object's environment, respectively. The meanings of T, t and p are the same as those defined in Definition 2.

If we mark the event set for an active object as E, the set of internal state of an active object as S_{in}, the set of external state of an active object as S_{out}, then we can define an EECA rule as:

Definition 4 An EECA rule of an active object is a triple-tuple $\{e, f(s_1, s_2), a\}$, where, $e\in E$, $s_1\in S_{in}$, $s_2\in S_{out}$, f is an assertion on s_1 and s_2, and a is an order set of the active object's actions.

Now we define the active object's semantic model as:

Definition 5 An active object class is a five-tuple $\{N, R, S_{in}, S_{out}, A\}$, where N is the global name system of an active object, which distributes every object of this class with a global unique identifier, R is the EECA set of the active object in this class, S_{in} and S_{out} are the sets of internal and external states of the active objects in this class respectively, and A is the set of actions of that the active object in this class can take.

Definition 6 An active object, which is an instance of an active object class, is an eight-tuple $\{I, R, T, S_{in}, S_{out}, A, T_a, a\}$, where I is global unique active object identifier, R is the EECA set of the active object, T is the trace of the active object's passed states, S_{in} and S_{out} are the sets of internal and external states of the active object respectively, A is the set of actions that the active object can take, T_a is the trace of the active object's passed actions, and a is the current action the

active object is taking.

Many active database researchers know that a difficult problem for ECA management in active database is the rule storm activated by recursion of the rule. From the state trace of EECA, we can see that EECA can easily avoid rule storm by detecting the active object's transaction trace.

INTERNET SERVICE EVOLUTION MODEL BASED ON ACTIVE OBJECT

Analysis results of (Chen *et al*., 2002) showed that the evolution of Internet Service can be categorized into four classes that are called "Evolution of Design", "Leak Package", "Load Balance", and "Failure Recovery". From the time scale, we can classify "Evolution of Design" and "Leak Package" as long term evolutions, and classify "Load Balance" and "Failure Recovery" as short term evolutions. For the long term evolutions, even though formal methods, such as use of software architecture, to study widely software evolution, there is still a wide gap from automatic analysis and processing. For short term evolutions, the automatic work may heavily depend on efforts of monitoring and extraction. From this point of view, we design the structure of the model for ISE as shown in Fig.2.

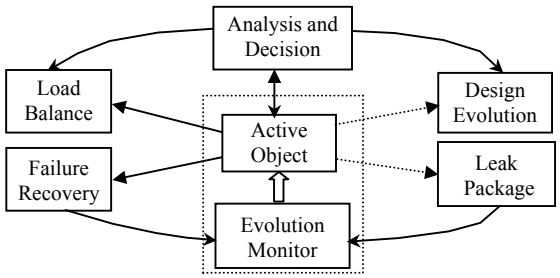

Fig.2 Model for Internet Service Evolution based on active object. The solid arrows describe the control and information flow, the dashed arrows indicate that there will be a new update version

The model has 7 components: "Evolution Monitor", "Active Object", "Analysis and Decision", "Load Balance", "Failure Recovery", "Design Evolution" and "Leak Package".

(1) "Evolution Monitor"

Based on the characteristics of Internet Service, "Evolution Monitor" detects and records all kinds of events of the system state and sends messages to the active object by events.

(2) "Active Object"

"Active Object" monitors state of system by getting messages from "Evolution Monitor". When an "Active Object" gets a message from "Evolution Monitor", it detects the current system condition. If the condition is true, the "Active Object" will take some predefined actions.

(3) "Analysis and Decision"

When a system encounters situations that cannot be solved automatically, "Analysis and Decision" component will get a notice from active object components and it will get control to help people do analysis and processing.

(4) "Load Balance"

This component is set for the adaptation and updating uncertainty of processes and loading tasks. The component schedules the internal resource and process tasks to assure the balance and quality of the processes.

(5) "Failure Recovery"

The causative reason for the existence of this component is that the Internet Service, which runs over the Internet, will inevitably encounter system failures such as hardware faults, network break-down, etc. This component will locate the position of the failures and try to extricate the system from these failures.

(6) "Design Evolution"

When there is a distinct architecture change coming to the system, this component will be in control. As we have mentioned above, most of this component's processes will be finished by the user, except that there exists a component which can be uploaded automatically.

(7) "Leak Package"

This component responds to the processes such as "defect finding" and "defect repairing". The processes of this component look like the "Design Evolution" and can be classified into two parts, which are called "automatically" and "manually".

IMPLEMENTATION OF EVOLUTION MONITOR AND ACTIVE OBJECT

As we discussed above in Section 3, "Evolution

Monitor" and "Active Object" are two key components for automatic ISE.

1. "Evolution Monitor"

The key to implement "Evolution Monitor" is to set up an effective mechanism for collecting data information on system running state. A good monitor should be easy to use, distribute, and maintain. In addition, it should have few fault reports. A normal way to monitor the current system state is to record the visited URL, IP and time stamp in a log file. Even though recording external events is really helpful for learning the running state of the Internet Service, it may be more important to know the internal state of the Internet Service for dealing with "Failure Recovery", "Leak Package" and "Load Balance". From this viewpoint, we design the "Evolution Monitor" as a component with structure shown in Fig.3.

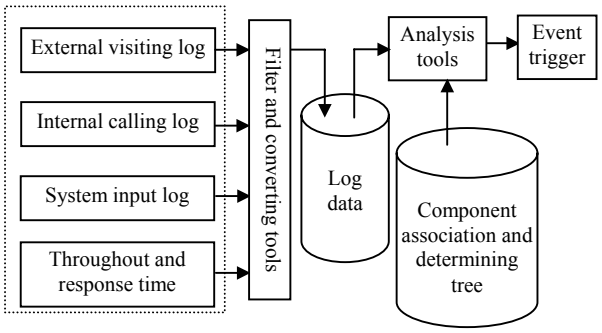

Fig.3 Structure of the monitor model

"External visiting log" is used to record the visited data from outside; "Internal calling log" is used to compute and record the calling trace among internal components; "System input log" is used to record all the input events and data from the external system; "Throughout and response time" records the system state and performance. "Component association and determining tree" are information on components calling path and system analysis; "Analysis tools" use system log and "Component association and determining tree" to analyze system state and trigger events for active objects to take actions when there is a requirement occurring. "Filter and converting tools" compose data exchange interface between recording tools and logging tools.

2. "Active Object"

Fig.4 shows that there are seven components in the active object system structure: "Meta data", "Rule base", "Data convertion" (there are two of this kind of components), "Analysis management", "Trigger management" and "Event queue". "Meta data" component provides meta specification for active object. "Rule base" component stores rule for active objects; there are two "Data convertion" components in an active object system, one component accepts triggering events from monitor system, and then converts and pushes the events into the active object's event queue. Another component sends control information to the analysis and evolution subsystem. It also accepts feedback information from the analysis and evolution subsystem to direct the maintenance work to "Rule base" component and "Meta data" component; "Analysis management" component monitors system state and triggers action events when there is a demand; "Trigger management" component schedules and distributes the process events to "Data convertion" component or "Event queue" component.

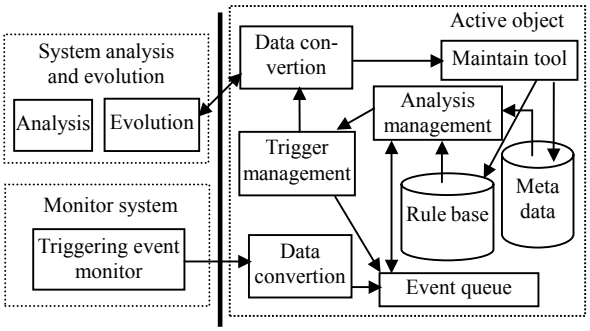

Fig.4 Structure of the active object

APPLICATION AND IMPLEMENTATION

We use the active object models introduced in Section 2 and the framework in Section 3 to develop a test platform for validating the ISE framework. The test platform is called "Travel Union" and the following subsections will introduce the key technologies for the platform's implementation.

The toolkits and environment for the development of the platform can be divided into two parts: "Client Toolkits" and "Server Toolkits".

1. Client Toolkits

Internet browser as IE 6.0 or Netscape 4.0.

2. Server Toolkits

(1) Two application servers with Microsoft Windows 2000 Advanced Server.

(2) Application logic server:

Operating system: Microsoft Windows 2000 Advanced Server;

Web server: Internet Information Server;

Application script server: Active Server Pages Server;

Component protocol and toolkit: DCOM and Visual C++.

(3) Database server:

Operating system: Microsoft Windows 2000 Advanced Server;

DBMS: SQL Server 2000;

Interface component: DCOM in Visual C++.

Our test platform also includes over 40 process components and 20 data tables. The data entities in "Travel Union" can be divided into 3 parts, which are shown in Fig.5.

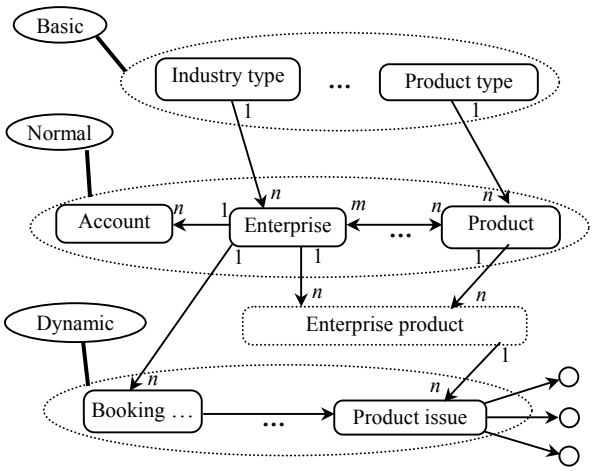

Fig.5 Entity-Relation model of Travel Union based on IPEIMC

The "Basic static information" includes code tables as "Product type" and "Industry type", etc. "Normal information" stores "Enterprise information", "Product information", and "User interface information", which is registered once and requested multiple times. "Dynamic information" stores information such as "Product price", "Booking information", and "New product list" and conducts exchanges between enterprises. Fig.5 is a simplified Entity-Relation example for Travel Union.

To monitor the running state and performance of the servers and components inside servers, the monitor keeps looking on the following aspects of the system.

(1) Use user's IP and cookies to monitor the visiting rate and performance load, when an abnormal heavy load is monitored, the monitor will trigger a heavily loaded event to active object.

(2) Use small plug-in modules for components to record the calling path among the Internet Service components. When abnormal calling paths and processes are found, the monitor needs to trigger an abnormal event to deal with.

(3) Use a scanner to monitor configuration and system states with the original template. To avoid false alarms, the scanner will use the following strategy to evaluate the system's true state.

```
T=t1;
for I:=1 to 3
begin
   sleep for T;
   access:=visit the service again;
   if (access=true) then  no error report and return;
   T:=T+datT;
end
report unaccess error;
```

When we develop the active object system, a difficult problem to solve is event communication over the Internet. To solve this problem, we set up an event channel mechanism based on CORBA. Each event channel in our development includes two entities: event consumer and event messenger.

The receiver works in a call back mode and the IDL specification of it is as follows:

```
struct Event {
   string event_topic;
   enum eventset event_conent;
   Time time;
   Location location;
};
interface Consumer {
   void push(in Event event);
   void disconnect(in string reason);
};
```

The messenger's IDL specification is as follows:

```
interface Notifier {
   void subscribe(in Consumer consumer,
```

```
        in string filteringcriteria);
      void unsubscribe(in Consumer consumer);
      void push(in Event event);
    };
```

The messenger's IDL specification is as follows (pseudo C++ code):

```
class My_Notifier {
 void subscribe(Consumer &consumer, const char *filtercriteria)
  {
      insert *consumer into <consumer_set> with filtercriteria;
  };
 void unsubscribe(Consumer &consumer) {
      remove *consumer from <consumer_set>;
  };
 void push(Event &event) {
    for each <consumer> in <consumer_set>
      if (event.topic matches consumer.filtercriteria)
        consumer.push(event);
  };
}
```

The following is the algorithm for a monitor who has found an abnormal and sent trigger event to the active object:

(1) With the help of "Data convertion" component, Monitor triggers a request for system evolution;

(2) "Data convertion" through "Resource interface" component and "Network Interface" component sends a request to the server in which the active objects are located in;

(3) "Data convertion" located in active object server first changes the receiving data to local data, then pushes them into the event queue;

(4) If event queue is not empty then "Analysis management" component executes Steps (5)~(8), else go to Step (9);

(5) Get an event from event queue and analyze it with Rule and Meta data;

(6) If the rule condition is true, go to Step (7), else go to Step (8);

(7) If the process calls to another active object, then push an event into event queue, else take action for evolution component;

(8) Go to Step (4) for next cycle;

(9) End of this process.

RESULTS AND CONCLUSION

Table 1 lists some of the test results using our test bed. Inside the row of "Evolution update", the table lists the numbers for the automatically downloading operations when there are new versions for system upgrade.

From Table 1, we can see that the model for ISE based on active objects is an effective mechanism for system evolution detection and automatic processing. Currently this work is still in its prototype phase. There are still false and miss reports. Our project group is working on these problems by maintaining the rule-base of active objects to adapt to the new challenge and requirement from the system evolution.

Table 1 Test results using the test bed

	Total	Normal	Found	Miss	Fault	Automation	Manual
Overload	50	10	30	10	3	23	14
Failure recovery	50	10	32	8	5	25	10
Evolution update	10	10	10	0	0	10	0

References

Campin, J., Paton, N.W., Williams, M.H., 1995. A structured specification of an active database system. *Information and Software Technology*, **37**(1):47-61. [doi:10.1016/0950-5849(94)00013-I]

CCF Project Team, 2000. CCF: A Framework for Collaborative Computing. *IEEE Internet Computing*, **4**(1):16-24. [doi:10.1109/4236.815845]

Chen, M., Kicman, E., Brewer, E., 2002. An Online Evolutionary Approach to Developing Internet Services. Http://www.cs.berkeley.edu/~brewer/papers/online-evolution-draft.pdf

Gadke, M., Graef, G., 2000. Development and Evolution of Web-applications Using the Web Composition Process Model. Http://www.teco.edu/~gaedke/paper/2000-www9-webe.pdf

Gong, B., Wang, S., 2001. Method getting model of MAS: supporting dynamic enterprise model. *Systems Engineering—Theory & Practice*, **21**(5):44-49 (in Chinese).

Hu, H., 2003. Platform on enterprise information management and cooperation based on Internet. *Systems Engineering—Theory & Practice*, **23**(5):31-35 (in Chinese).

Hu, H., Yu, H., 2002. A global web workflow system based on active object. *Journal of Software*, **13**(8):1672-1677 (in Chinese).

Ingham, D.B., Caughey, S.J., Little, M.C., 1997. Supporting highly manageable web services. *Computer Networks and ISDN Systems*, **29**(8-13):1405-1416. [doi:10.1016/S0169-7552(97)00044-5]

Jiang, G., Fan, Y., Wu, C., 2002. Dynamic collaboration and enterprise agility. *Systems Engineering—Theory & Practice*, **22**(1):71-74 (in Chinese).

Kicman, E., Fox, A., 2004. Detecting Application-level Failures in Component-based Internet Services. Http://www.stanford.edu/~emrek/pubs/anomaly-2.pdf

Kicman, E., Wang, Y., 2005. Discovering Correctness Constraints for Self-management of System Configuration. Http://www.stanford.edu/~emrek/pubs/glean.pdf

Kim, Y., 2000. WW-flow: Web-based Workflow Management with Runtime Encapsulation. *IEEE Internet Computing*, **4**(3):55-64. [doi:10.1109/4236.845391]

Kirda, E., Jazayeri, M., Kerer, C., 2001. Experiences in engineering flexible web services. *IEEE Multimedia*, **8**(1):58-65. [doi:10.1109/93.923954]

Lee, T.B., 1994. World-wide web. *Communication of ACM*, **37**(7):76-82.