Design of an Aspect-Oriented Software Architecture for Web-based Information System

Atsushi Kumazaki

Dept. Info. Sys. & Quant. Sci., Nanzan Univ.
18 Yamazato, Showa, Nagoya, Japan 466-8673
d00bb002@nanzan-u.ac.jp

Masami Noro, Han-Myung Chang
Yoshinari Hachisu
Dept. Info. & Telecomm. Eng., Nanzan Univ.
27 Seirei, Seto, Japan 489-0863
{yoshie, chang, hachisu}@nanzan-u.ac.jp

ABSTRACT

This paper describes an aspect-oriented software architecture for Web-based information systems(WIS), which is aspect-oriented version of the architecture we previously constructed. The key issues for developing WIS, nowadays, are how to select and coordinate existing Web-based technologies such as CGI scripts, applets, PHPs, servlets and so on. There is unfortunately a bit chaos and confusion on the coordination and they enforce troublesome development. An aspect-oriented software architecture implies a concurrent software process. The architecture is a guide map for the selection and the coordination. It eases the development of WIS as a result.

Keywords

Software Architecture, Aspect-Oriented, Development Process, Web-based Technology

1 Introduction

Software engineering community becomes to put much more emphasis on the development of Web-based information systems(WIS)[16, 18, 20]. The key issue for developing WIS becomes how to select and coordinate existing Web-based technologies such as CGI scripts, applets, PHPs, servlets and so on rather than how to define requirements and to design it. There is unfortunately a bit chaos and confusion on the coordination since no guidelines are suggested for the selection and coordination. Consequently, the chaos and confusion enforce cumbersome development on developers. We constructed an aspect-oriented software architecture for WIS to solve the problem.

The aspect-oriented software architecture described here for WIS is aspect-oriented version of the architecture we previously constructed[9]. We have tackled several software engineering problems with software architectures[5, 14]. In these days, aspect-oriented programming is recognized as another breakthrough to the problems[3]. We applied an aspect-oriented concept to a software architecture.

We have demonstrated with several projects that a software architecture is not just a product model but implies a software process[9, 10]. In the projects, we found that a software architecture consists of a set of aspects. A software architecture implies a partial order of developing aspects, and an aspect implies a partial order of developing components in that aspect. As a result, an aspect-oriented software architecture implies a concurrent software process.

There are several related research projects on aspect-oriented software architectures[12] and aspect-oriented application frameworks[1, 11]. Navasa and Nakajima put much emphasis on construction process of architecture or framework. Constantinides gives an idea how to implement an application framework with a design pattern (Factory Method)[1]. They just talk about product view of software development, but we also introduces process view of software development.

2 Software Architecture

We realized that a software architecture is not just a product model but also a container including a process planning scenario[9, 10]. The insight we gained meets the definition by the SEI discussion group in '94 of a software architecture that "the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."

The software architecture includes components from the software model, the implementation stage, and the design result. Fig.1 shows our software architecture including a process planning scenario.

The development process, which the software architecture shown in Fig.1 indicates, can be summarized as follows.

- Extract implementation-level component and plan to select the appropriate class or component library.
- Implement classes and their hierarchy for the design-level component.
- Carry out the design and implementation for a component from the software model.

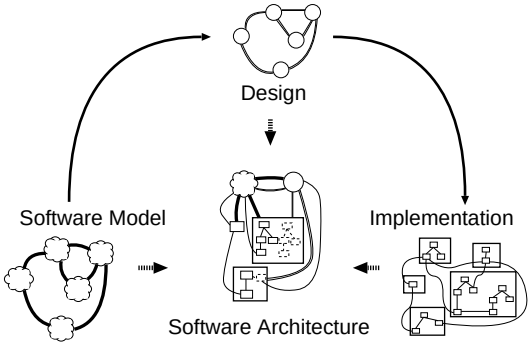

Fig. 1: Software Architecture

2.1 Aspect, Architecture and Process

We consider that an aspect-oriented software architecture is composed of a set of aspects. An aspect is a composite component. In this sense, if we apply a component-connector model to an aspect-oriented software architecture, an aspect is a component, and a join point is a connector. Fig.2 cartoons an example of our aspect-oriented software architecture.

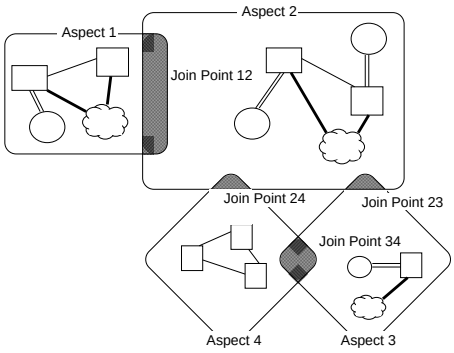

Fig. 2: Aspect-Oriented Software Architecture

2.2 Process Derivation Rule

Hetero-links (connectors in different types) in an aspect imply a software process for the development of an aspect, and the relationship among aspects defines a partial order of aspect development.

For Aspects If two aspects crosscut each other, inter-aspect protocol on the join point which associates the aspects must be defined first. Fig.3 shows a development process derived from crosscut aspects. We borrow the activity diagram in RUP to describe a process.

For Components Here we assume that there are tree levels of abstraction: model, design, and implementation. We can derive a development process for each abstraction level component, shown in Fig.4.

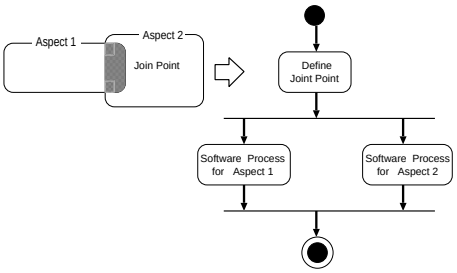

Fig. 3: Process-Derivation Rule for Aspects

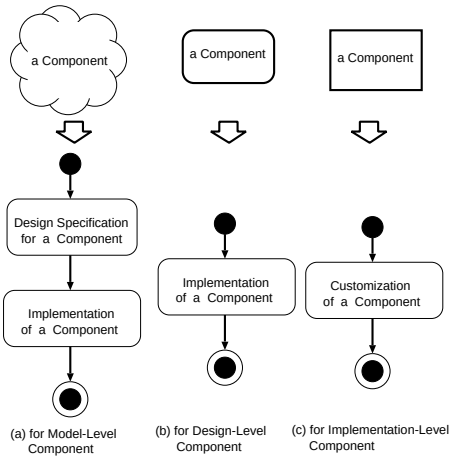

Fig. 4: Process-Derivation Rule for Components

2.3 Organization of Processes

There are nine $(3 \times 2 + 3)$ use-relations (send-message-relations) among components in three levels of abstraction. Three relations are ones for the same level of abstraction. Other six are for components from different level of abstraction. Fig.5 shows an example of the software process obtained from a relation in which a model-level component uses another model-level component. Software processes obtained from other relations are omitted to save the space [13].

In an aspect-oriented software architecture, software processes for the development of aspects are combined into a software process for the architecture. Definitions for inter-aspect protocols (join points) precede the processes for the aspects. If a software architecture consists of three aspects crosscutting one another and each aspect is composed of components as in Fig.6 (a), the software process will be one shown in (b).

3 Aspect-Oriented Software Architecture for WIS

We show an aspect-oriented software architecture which we have constructed for WIS. In our previous software architecture, we modeled a WIS as a set of an object-oriented database application and an interac-

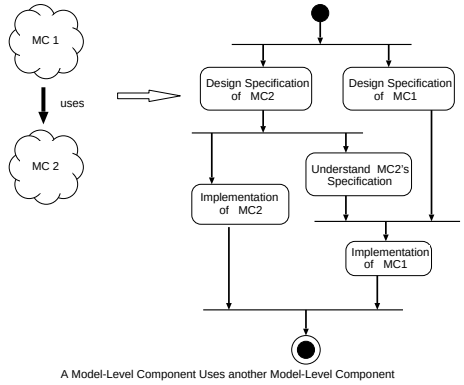

Fig. 5: Software Process Organization Rule

tive application. As a result, we integrated the three-tier architecture[2] and the MVC architecture[6], and adopted them for WIS.

Through the construction of a software architecture for WIS, we realized that several concerns crosscut more than one layer. They are controller, application logic, and so on. In the following sections, we treat those concerns as well as component of dominant decomposition as aspects in the aspect-oriented software architecture for WIS.

3.1 Three Views of Software Architecture

To represent a software process in a software architecture, we describe a software architecture from the following three views: abstract view, concrete view and process view. It is an improved and revised version of Kruchten's definition[8]. Kruchten's software architecture composed of the same abstraction level components, but our software architecture composed of the different abstraction level components.

We also use these three views for defining an aspect-oriented software architecture. The abstract view corresponds to component-connector model. The concrete view identifies abstraction level of components. Process view holds the process that shows the partial order of component development.

3.2 Aspects in WIS

Here, we describe aspects in WIS. The following concerns are recognized through observing the construction of the software architecture for WIS.

– Controller :

  This concern is about MVC's Controller. A WIS is an interactive application. The Controller handles user's inputs.

– View :

  This concern is about MVC's View.

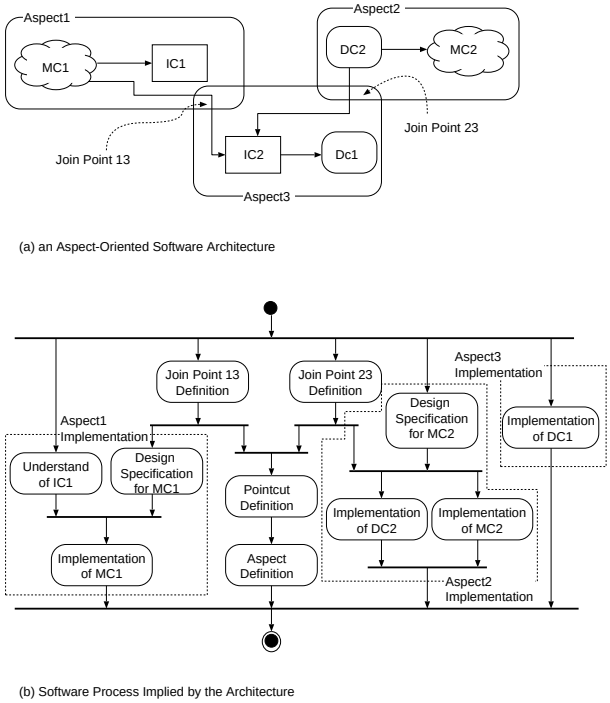

Fig. 6: Aspect-Oriented Software Architecture and its Development Process

– Application Logic :

  A WIS is basically a database application. This concern is about how to implement application logic including access to the database.

– Platform :

  When we develop a WIS, we use Web-based technologies, such as CGIs. These technologies depend on a platform, such as a server's OS. This concern is related to a server's platform.

– Page :

  WIS present information to users. This concern is about how to implement these information. Generally, we implement information using HTML.

– Efficiency :

  A WIS increase the server load. This concern is about how to decrease the server load.

– Security :

  User's information, such as password and so on, flow on the internet. This concern is about how to protect those information.

Fig.7 shows these concerns, aspects, and their relationships in a WIS. As in the figure, an aspect is a container to implement several concerns (a single aspect may be used to realize a single concern.) Concerns are collected into a single aspect if concerns do not crosscut one another. Concerns in an aspect are related one another.

To relate means that one of components which realize a concern uses (sends a message) to one of other components implementing another concern, or vice versa. An arrow in the figure means that the component(s) implementing a concern which is a source of an arrow uses one of the components which realize a concern at the destination of the arrow.

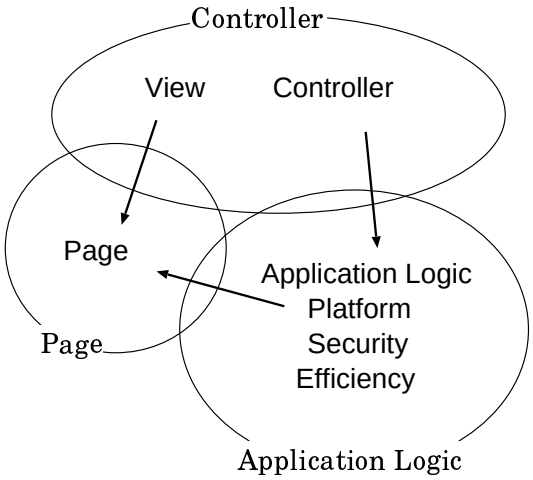

Fig. 7: Concerns, Aspects, and their Relationships in a WIS

3.3 Abstract View of WIS Architecture

Our aspect-oriented software architecture for WIS is aspect-oriented version of the architecture we previously constructed[9]. We borrow portions of the previous architecture to draw the aspect-oriented architecture.

Fig.8 outlines our aspect-oriented architecture for WIS. `Controller` in the figure handles user inputs. `Application` implements application logic, such as accessing the database. `OO_DB_Proxy` implements object-oriented interface for accessing the database. `Model` is a model for displaying information for users, and `View` displays. `Model`, `View` and `Controller` are components for MVC architecture. Since WISs are interactive applications, applying the MVC architecture is a natural idea.

3.4 Concrete View of WIS Architecture

The concrete view defines abstraction levels of components. The application logic aspect includes application dependent objects. Since they are heavily application-dependent, these components are model-level components.

The controller aspect includes components for implementing a user interface. Since a user interface for WIS is a Web browser, these components are implementation-level components, that is a Web browser.

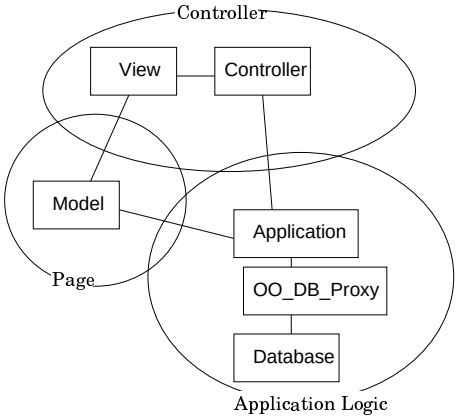

Fig. 8: Abstract View

If a Web browser is a user interface, the page aspect constructed by HTML. Developers implement `Model` using HTML. This component is a design-level component.

These discussions are summarized in Fig.9. It shows abstraction levels to the components and arrows which represent development process.

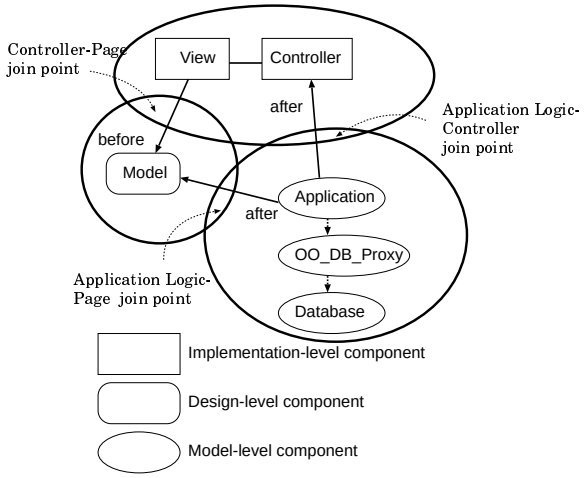

Fig. 9: Concrete View

3.5 Process View of WIS Architecture

Fig.10 shows a development process which can be drawn from the concrete view.

4 Case Studies

We have developed WISs based on this software architecture for discussing which Web-based technology is useful in which case. Web-based technologies that we use for developing WISs are CGI, Java, and PHP. These are abstracted in Fig.11

4.1 CGI

When we develop WIS using CGI, we implement application logic by CGI programs or scripts. Since CGI

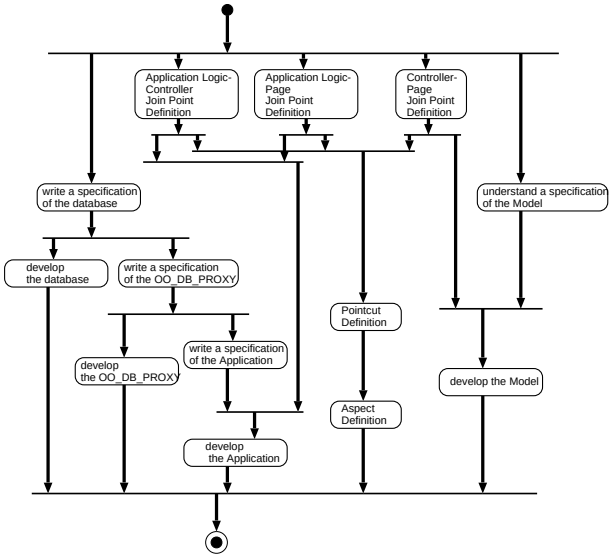

Fig. 10: Process View

programs or scripts can be written by various programming languages, we can write programs or scripts using familiar programming language.

We implement `OO_DB_Proxy` using libraries provided for each programming languages. If it is no library, developer implement database accessing.

4.2 Java

In the case of Java, Java Servlet mediate between a HTML document and application logic. We implement application logic using Java, and Java Servlet is the mediator[4] between a HTML document and a Java program. In this case, it is proper to construct model using JSP and to implement application logic by Java Servlet. The role of each Web-based technologies becomes clear.

JDBC(TM) is familiar interfaces of Java programs and DBMS. We implement `OO_DB_Proxy` using JDBC(TM).

4.3 PHP

PHP is a programming language which can be written in HTML documents. Application logic is implemented by PHP, and cooperate with HTML documents. Further, PHP has interfaces to various DBMSs.

5 Discussions

We discuss which Web-based technology is useful in which case.

(1) Efficiency & Security & Platform

These are concerns which are included by `Application Logic`. When we select a Web-based technology and implement `Application`, we must consider these concerns.

For efficiency, selecting Java Servlet or PHP is better than CGI. CGI create new heavy-weight process per

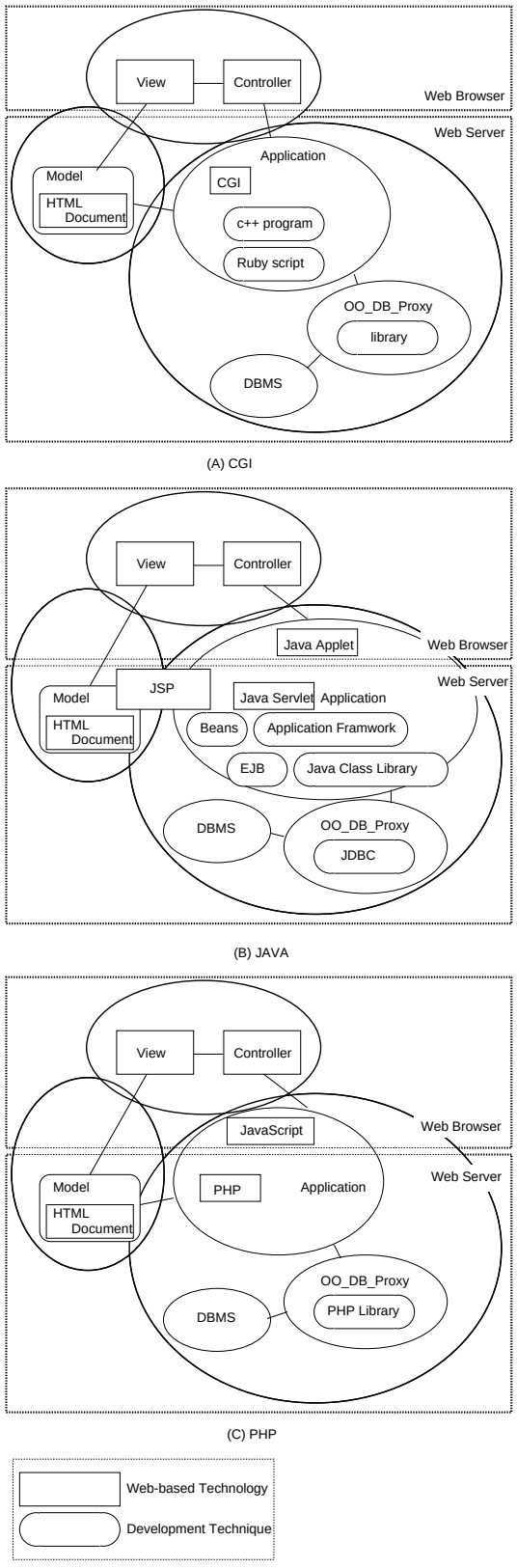

Fig. 11: WIS Implementation

HTTP request. But Java Servlet and PHP are processed by single process, a Java Virtual Machine or a PHP interpreter.

For security, it is necessary to authenticate users for on-line trade[17] or internet banking[15], etc. This is a typical case where internal transaction must be secret for users. In this case, these transaction must be implemented by server-side Web-based technologies, such as CGI or Java Servlet, etc.

`Platform` relate to server's OS. If Windows run on the server, we can select ASP(Active Server Page) instead of JSP. But ASP is essentially the same as JSP. There is no reason that we must select ASP instead of JSP.

(2) User Interface & **Animation**

The User Interface which cannot be implemented by only HTML is implemented by client-side Web-based technologies, such as Java Applet or JavaScript. Application logic which is no need to access the database, such as animation[19], can be implemented by these technologies, too. These technologies are used to make good server-side technologies.

(3) Flexibility & **Reusability**

We use object-oriented technologies to append the flexibility and the reusability to WISs. In the case of CGI, CGI programs or scripts are generally written by C or Perl. But these programming languages are procedure-oriented programming languages. We prefer C++ or Ruby. These programming languages are object-oriented version of C and Perl.

In the case of Java, since Java is a object-oriented programming language, we use object-oriented technology naturally. Beans, EJB, and Java's class library are already implemented. We can use these items to develop WISs.

In the case of PHP, PHP is a procedure-oriented programming language. But we can program in object-oriented way using only object-type of PHP. It is possible to use object-oriented technologies.

6 Conclusion

We constructed an aspect-oriented software architecture for WIS. This software architecture implies concurrent development process for WIS, it can help us to select and coordinate Web-based technologies. As a result, it eases the development of WIS. Actually, we have developed WISs, and recognized this software architecture is useful for developing WIS.

REFERENCES

1. C. Constantinides, A. Bader, T. Elrad and M. Fayad "Designing an Aspect-Oriented Framework in an Object-Oriented Environment," *Computing Surveys 32, 41*, 2000.
2. J. Donovan, *Business Re-Engineering with Information Technology*, Prentice Hall PTR, 1994.
3. T. Elrad, R. E. Filman, A. Bader, Eds., *Special Issue on Aspect Oriented Programming*, CACM, Vol. 44, 2001.
4. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1995.
5. D. Garlan and D. E. Perry Eds., *Special Issues on Software Architecture : IEEE Transaction on Software Enginnering, Vol. 21, No. 4*, 1995.
6. A. Goldberg, *Smalltalk-80 : the interactive programming environment*. Addison-Wesley, 1984
7. G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *J. of Object-Oriented Program*, Vol. 1, No. 3, pp.22-49, Aug./Sep. 1988.
8. P. B. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, Vol. 12, No. 6, pp.42-50, Nov. 1995.
9. A. Kumazaki, et. al., "A Software Architecture for Web-based Information Systems," (in Japanese) in *Proc. of IPSJ OO Symposium, pp.93-96, 2002.*
10. A. Kumazaki, et. al., "An application Framework for TCP/IP Applications," in *Proc. of COMPSAC, pp.627-634, 2002.*
11. "Separation of Concerns in Early Stage of Framework Development," Workshop on Multidimentional Separation fo Concerns, *OOPSLA2001.*
12. A. Navasa, *et al.*, "Aspect Oriented Software Architecture: a Structural Perspective," http://wwwhome.cs.utwente.nl/~bedir/Research.htm.
13. M. Noro, et, al., "On Aspect-Oriented Software Architecture: It Implies a Process as Well as a Product," (to appear) in *Proceedings of APSEC 2002.*
14. M. Shaw, and D. Garlan, *Software Architecture, Perspective on an Emerging Discipline*, Prentice Hall, 1996.
15. e-bank, http://www.ebank.co.jp.
16. YAHOO! JAPAN, http://www.yahoo.co.jp.
17. YAHOO! Auctions, http://auctions.yahoo.co.jp.
18. YAHOO! MESSAGE BOARDS, http://messages.yahoo.co.jp.
19. YAHOO! GAMES, http://games.yahoo.co.jp.
20. goo, http://www.goo.ne.jp.