Short Paper

DESIGN AND IMPLEMENTATION OF A FLEXIBLE FUNCTION ACCESS CONTROL TOOL FOR WEB APPLICATIONS

Kung Chen* and Chih-Shang Chang

ABSTRACT

This paper presents a flexible and practical tool, F-menugen, which supports function access control on the presentation tier by restricting user menus to functions that a user's access privileges permit. The menu structure and rules governing the functions accessible to a user are specified declaratively in an XML configuration file; the rules are based on user profiles, application-specific requirements, and certain contextual information. After user login, the tool will generate a tailored function menu according to the user's access privileges. Future changes of access control rules can also be effectively realized by modifying the configuration file without actual coding.

Key Words: access control, web applications, XML.

I. INTRODUCTION

There have been many mechanisms and frameworks developed to help Web application developers implement access control in an effective and modular manner, among which the Java Authentication and Authorization Services (JAAS, 2002) of J2EE is a well-known and representative work. While effective for preventing unauthorized access, solutions such as JAAS also have some shortcomings. Specifically, such frameworks conduct the access control checks along with the invocation of an application function. In terms of the well-accepted three-tier architecture of Web applications (Curphey *et al.*, 2002), these checks are performed on the application tier (the other two are presentation tier and data tier). As a result, users will see functions which they are not allowed to execute. Furthermore, any attempted access to unauthorized functions, whether intentionally or unintentionally, will lead to an application tier security check and incur a certain amount of runtime overhead. Finally, users may feel annoyed or confused about getting access violation messages from time to time.

Indeed, in many situations, it is possible to determine whether a particular function should be authorized to a user without actually having to try to perform it. As for invalid operations, users should not be able to see them on the function menu in the first place, and the application tier check should be reserved to more fine-grained content-based access control requirements. Therefore, we claim that a two-stage approach is a better way to organize the task of access control enforcement. The first stage is conducted on the presentation tier by restricting user menus to only functions granted by a user's access privileges; and the second stage on the application tier through mechanisms such as JAAS. By dividing the enforcement of access control into two stages, we can overcome the shortcomings described above yet retain the same security derived from a conventional approach to implementing access control.

To realize this approach, we need to provide a proper user interface tool for Web application developers. Mainstream tools for rendering Web application user interface, such as ASP (Microsoft,

*Corresponding author. (Email: chenk@sc.nccu.edu.tw)

K. Chen is with the Department of Computer Science, National Chengchi Unicersity, Taipei 116, Taiwan.

C. S. Chang is with the Department of Information Management, Tung Nan Institute of Technology, Taipei 116, Taiwan

2003) and JSP (JSP, 2004), tend to encourage a bad style of mixing application logic with presentation logic, which makes the resulting code not only hard to change but also prone to errors (Curphey *et al.*, 2002). It would be even worse if accessibility-based control code was required in rendering user interface and function menus. Hence this paper presents a flexible and practical tool, *F-menugen*, that restricts user menus to functions that a user's current access-privileges permit, and can thus support function access control on the presentation tier and overcome those shortcomings.

In our approach, the menu structure and rules governing the functions accessible to a user are specified declaratively in an XML configuration file; the rules are based on user attributes, application-specific parameters, and certain contextual information. After verifying a user's identity, our framework will trigger F-menugen to generate a tailored function menu according to the user's access privileges. Developers are freed of the tedious and error-prone task of encoding function access control in presentation code. Furthermore, if the access control rules or a user's privileges are changed, the user interface will automatically adapt to provide only the permitted functions to a user, thus eliminating the need to re-code the user interface.

The remainder of the paper is organized as follows. Section II reviews related work. Section III provides an overview of our approach. Section IV describes in detail our design and implementation of the accessibility-based menu generator. Section V concludes the paper.

II. RELATED WORK

Most access control frameworks for distributed client/server systems focus on enforcing access control on server-side components, which belong to the application and data tiers (Beznosov and Deng, 2002), (Kouadri Mostefaoui and Brezillon, 2003). Our work bears a closer relationship with that of Goodwin *et al* of IBM (Goodwin *et al.*, 2002). They also adopt an MVC-like architecture for Web applications and use an interaction controller to coordinate the backend activities, encapsulated by commands. In their framework, there are actually two levels of authorization: command-level and resource-level. The command-level access control is included as part of the command dispatching process and the resource-level control is performed within each command. Both are conducted on the application tier by a centralized authorization manager.

In contrast, our framework focuses on performing the command access control check on the presentation tier by restricting a user's menu to only functions accessible to him. By doing this, we believe that we can reduce the load of runtime authorization checks and make the application more user-friendly. In this sense, F-menugen can be viewed as providing some optimization for the command-level access control by moving it to the presentation tier.

III. OVERVIEW OF OUR APPROACH

The section outlines the main ideas behind our approach. First, it describes how we model access control requirements into rules on two different levels: function and data. Second, it highlights how our tool can effectively enforce function level access control rules for Web applications in a declarative manner.

1. Access Control Modeling and Rules

Access control, also known as authorization, is the process by which users, after being identified, are granted certain privileges with regard to a system's information, functions, or resources. The step to identify a user is usually called authentication. Username and Password check are the most basic form of user authentication. Authentication and authorization are the basic security requirements of any enterprise applications.

This paper focuses on application-level access control requirements that must consider factors specific to its users and current application state. We adopt a simple yet generic modeling approach that can support a wide range of access control requirements. Specifically, we model the interaction between a user and a Web application as a sequence of access tuples of three elements: (*user, function, data*), indicating a user's request to execute a function on a specific data object(s). The access control rules of an application determine which access tuples are allowed and which must be denied.

Clearly, all application functions that need to be protected from unauthorized access must be associated with some access control rule. Hence we associate an access constraint with each function to protect. A constraint is a Boolean expression that specifies under what conditions the designated function should be allowed to run. We take an object-based approach to specify access constraints. In particular, we assume five generic objects: *User*, *Form*, *Data*, *Time*, and *Param*, with various attributes that the constraint expression can refer to. In practice, the specific set of attributes for each object depends on an individual application's needs. Conceptually, the Form object and the Data object serve as the input and output of a function to execute, respectively. Typical attributes for the User object include user's name, title, department, and roles in an organization.

Both the Param object and the Time object are global to an application. The former stores various application-specific parameters related to access control, while the latter provides methods to retrieve the date and time values of any attempted access. Using these objects, we can accommodate a wide range of access control requirements.

For example, rule R1 below dictates that, in an order management system, the order deletion function is authorized only to users whose title is "Sales manager".

R1: {deleteOrder, User.title = "SalesManager"}

Furthermore, security administrators can use the Time and Param objects to express many access control requirements involving time and locations. For instance, certain functions are accessible only during working days and from specific machines. We address such requirements by making the definitions of working days and privileged machine address the attributes of the Param object. For example, rule R2 below refines R1 and dictates that the batch printing function is accessible to sales managers at headquarters only during working days and from dedicated machines.

R2: {batchPrint, User.title = "SalesManager"
and User.officeLocation = "HQ"
and Time.getDay() in Param.workingDays
and User.machineIP in Param.privilegedMachineIP}

Rules like R1 and R2 above which determine the accessibility of application functions based on mainly user profiles and some contextual information are called function level rules. In addition, there are also rules that require user input or data content information to express their constraints. Such rules are called data level rules. In our modeling scheme, their constraints will refer to the Form or the Data objects. The attributes of the Form object include the arguments passed to the protected function, while the attributes of the Data object hold the data fields resulting from executing the designated function. Rule R3 and R4 below are typical examples of data level access control rules in an order management system.

R3: {createOrder, Form.totalAmount < 100000 or User.type ="VIP"}
R4: {viewOrders, User.department = Data.creatorDept)}

Here rule R3 specifies that only VIP users can create orders whose total amount is greater than 10000, and R4 requires that a user can view only orders created by users of the same department.

Obviously, from an implementation viewpoint, rules like R1 and R2 can be enforced before the designated functions are requested by any users, since information regarding a user's attributes and access time and location are available to the application immediately after the user has passed the identity check. By contrast, rules like R3 and R4 require user input or data contents which are only available when executing the associated functions. Hence, we should handle these two kinds of access control rules differently. Indeed, for functions whose access constraints are mainly dependent on user information, we can make their access control decisions once we get the requesting user's identity information; and the earliest time for doing it is when we are preparing the function menu that will be rendered to an authenticated user. In other words, if we know that a user's access rights do not satisfy a function's constraint, then we can exclude it from the user's function menu. This motivates us to develop a flexible menu control tool.

2. Web Application Architecture and Function Access Control

We follow the popular three-tier architectural principle and divide a Web application into three logical tiers: presentation, application, and data tiers (Curphey *et al.*, 2002). Furthermore, as our goal is to develop a practical and flexible tool that can enforce function access control on the presentation tier, we must carefully design our tool to integrate well with mainstream presentation frameworks for Web applications.

Many architectural frameworks have been proposed for structuring the presentation tier, and most of them have a common basis, namely the Model-View-Controller (MVC) design pattern (Gamma *et al.*, 1995). The popular open source Struts framework (Apache, 2001) is a representative example of such MVC-based frameworks for Web applications. The MVC pattern essentially separates an application into three parts: a Model, a View, and a Controller. The model components encapsulate the application components in the business logic and data tiers. The view components are those pieces of an application that display the information provided by model components and accept input. These view components are generally built using page-based scripting tools such as ASP and JSP. The controller is a special program unit that coordinates all the activities of the model and view components. It acts as a central point of control within an application.

Figure 1 illustrates the structure of an MVC-based Web application, enhanced with F-menugen. Basically,

every user request from the browser will be dispatched to a model element by the controller according to some pre-defined mapping rules, usually specified in a configuration file. These model elements are responsible for serving the requests or passing them to the proper business logic components, and for returning the correct view element that the controller should forward to after finishing the user request. This view forwarding is also based on some pre-defined mapping rules. After receiving a forwarding request, the controller will invoke the designated view element to display the results. Operating this way, the model elements and the view elements are adequately de-coupled so that the business logic components can be largely independent of the user interface implementation.

We have designed the menu generator to integrate well with MVC-based Web applications. Specifically, for access control purposes, all applications employ an authentication module to identify their users. This module can be viewed as a special model element of an MVC-based application. Our F-menugen is the view component associated with this authentication module. The standard user scenario is as follows. When an application is started, the user will be prompted for identification information by the authentication module. Then the controller will dispatch user input to the authentication module for user identity check. If successful, control will be transferred to F-mentugen via the controller. Furthermore, all information regarding the underlying application's menu structure and user access control rules is specified in an XML-based configuration file. When invoked, F-menugen will interpret it along with user profiles to generate an accessibility-based function menu for user viewing, thus achieving the required function access control in a declarative manner.

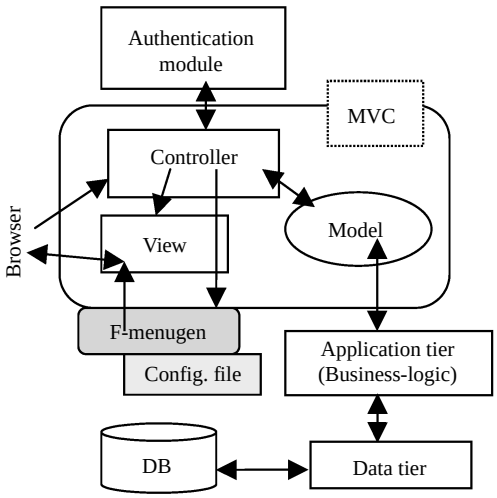

Fig. 1 MVC-based Web application architecture

IV. THE DESIGN AND IMPLEMENTATION OF F-MENUGEN

This section presents the design and implementation of F-menugen. We focus on the major design tasks of menu configuration and access control rules. In particular, we explain how we organize the functions of an application into a hierarchical menu that allows us to specify the access control rules for them in a flexible yet scalable manner. Besides, we also describe the implementation scheme and code structure of the major components of F-menugen.

1. Function Grouping and Menu Configuration

Conceptually, all application functions that need to be protected can be associated with an access control rule. However, in reality, it is not practical to specify these access control rules on an individual function basis, for that would incur a huge amount of administration effort and would not scale up well. Instead, it is very common, in practice, to group related functions and consider their access control together. Therefore, to keep a proper balance between flexibility and scalability, our scheme requires that all the functions of an application are organized into a hierarchy and represented by a tree-based menu to a user. A security administrator can attach one access control rule to an individual function or a group of functions organized under the same ancestor node. The tree structure of an application's functions and the associated access control rules are both specified in an XML-based menu configuration file. Listing 1 shows the XML Document Type Definitions (DTD) for menu configuration files.

The *MenuTree* is the root element of the menu. Under it, there is one mandatory *ApplicationSystem* element and one optional *GlobalDeclaration* element. If present, the GlobalDeclaration element defines global variables that can be referenced in access control rules for this menu. For example, the application-specific parameters mentioned earlier will be packaged to an object and declared as such a global variable for specifying access constraints.

Our menu configuration models a four level hierarchy comprising *ApplicationSystem, Application, Function Group, and Function*. There must be only one ApplicationSystem element, while we can define multiple Application elements under it. In turn, we can define multiple FunctionGroups and/or Functions under each Application element. Functions are the leaf nodes in the menu tree, yet FunctionGroups can be nested to support a deep hierarchy of functions.

Besides name, both Function Group and Function element have the following attributes:

- *href*: It specifies the URL path for the corresponding backend function the menu item links to. Namely, it is a reference to the function to be performed when the menu item is chosen.
- *target*: It specifies into which window frame to render the results returned by the function linked with *href*. In our design, we assume the browser window is partitioned into at least two frames. One is the menu frame and the other is a working frame for displaying user input and results.

Application, Function Group, and Function elements all have *Display* elements. The Display element has a *DisplayText* element, which defines what is rendered visually for the menu item. It has two attributes:

- type: It specifies the display style of a menu item. The possible values for this attribute are text and image. If not defined, we use text as default.
- locale: If this attribute is defined, it specifies under which local language encoding this menu item should be displayed.

Listing 1: menu.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT MenuTree
    (GlobalDeclaration?, ApplicationSystem) >
<!ELEMENT ApplicationSystem (Application*) >
<!ELEMENT Application
    (Display, (FunctionGroup |
    Function)*, Rules) >
<!ELEMENT FunctionGroup
    (Display, (FunctionGroup | Function)*) >
<!ELEMENT Function (Display) >
<!ELEMENT Rules (Rule*) >
<!ELEMENT Display (DisplayText*)>
<!ELEMENT GlobalDeclaration (#PCDATA) >
<!ELEMENT DisplayText (#PCDATA) >
<!ELEMENT Rule (#PCDATA) >
<!ATTLIST MenuTree name #REQUIRED>
<!ATTLIST Application name #REQUIRED>
<!ATTLIST FunctionGroup name #REQUIRED>
<!ATTLIST FunctionGroup href #IMPLIED>
<!ATTLIST FunctionGroup target #IMPLIED>
<!ATTLIST Function name #REQUIRED>
<!ATTLIST Function href # REQUIRED>
<!ATTLIST Function target #IMPLIED>
<!ATTLIST DisplayText locale #IMPLIED>
<!ATTLIST DisplayText type #IMPLIED>
<!ATTLIST Rule path #REQUIRED>
```

2. Access Control Rule Specification

The rules that control the accessibility of functions are collectively stated under the Rules element, which is a child element of the Application element. A Rules element comprises multiple *Rule* elements, each of which has two parts: a function *path* and a *constraint*. The function path refers to a single function or a group of functions organized as a tree. The constraint is a Boolean expression which evaluates true or false. False constraints imply inaccessibility of the specified function or functions.

To make the rule specification task easier for Web application developers, we use the popular Web scripting language, JavaScript (Mozilla, 2000), for specifying access constraints. As noted earlier, the function access constraints may refer to user profiles, application-specific parameters and possibly some contextual information, modeled as generic objects. In our scheme, these generic objects will be implemented as three JavaScript objects: *user*, *param*, and *time*. Besides, we also provide several utility functions, such as *contains* and *equals*, for specifying access control constraints.

With these objects and utility functions, we can easily present the access control constraints as a JavaScript expression. For example, rule R2 given in Section III.1 will be expressed as the following form in our configuration file (the "&&" is the logical "and" operator of JavaScript).

```
<rule path="/OrderMgmt/batchPrint">
        equals(user.getProperty("title"),
        "SalesManager")
        && equals(user.getProperty
        ("officeLocation"), "HQ")
        && contains(param.getProperty
        ("workingDays"), time.getProperty
        ("day")
        && contains(param.getProperty
        ("privilegedMachineIP"),
        user.getProperty("machineIP")) ;
</rule>
```

Here the path attribute states that the function to protect, *batchPrint*, is under the application called *OrderMgmt*, and the constraint is simply a direct encoding of R2 in JavaScript.

As mentioned above, all functions are organized into a tree-based hierarchy to facilitate menu organization. This way of organizing function menus is also fully utilized in specifying access control rules. Specifically, a rule can be attached to a specific function or a function group, and the accessibility of an element (Function or FunctionGroup) is defined in a *cascading* manner:

- If the accessibility of the element is explicitly defined in the configuration, use the defined accessibility;
- Otherwise uses the accessibility of the nearest

ancestor that is explicitly set,
- If no ancestor has accessibility explicitly defined, make the element always accessible.

For example, the following two rules specify that all functions in function group FG1 are accessible to sales representatives, except function batchPrint, which is only authorized to sales managers.

```
<rule path="/OrderMgmt/FG1">
      contains(user.getProperty("title"),
      "Sales") ;
</rule>
<rule path="/OrderMgmt/FG1/batchPrint">
      contains(user.getProperty("title"),
      "SalesManger") ;
</rule>
```

Moreover, by combining JavaScript logical primitives and our utility functions, we can easily specify many subtle constraints in a concise manner. For example, the following rule demands that, as long as the application system is not under operation, all users, except those whose types contain only Guest, are allowed to access the functions under function group TestingFG ("!" is the logical negation and "||" is the logical "or" of JavaScript).

```
<rule path="/myApp/TestingFG">
      !(containsOnly(user.getProperty("type"),
      "Guest")
      ||equals(param.getProperty("applica-
      tionStage"), "Operation" ) );
</rule>
```

Finally, since all the rules are specified in an XML configuration file, it is very easy to modify them and there is no need to re-compile any parts of F-menugen to accommodate the changes.

3. Implementation Scheme and Code Structure

F-menugen works in collaboration with the user authentication module to perform the task of generating an accessibility-based function menu. It consists of three major components: *menu rendering*, *menu service*, and *rule engine*. Fig. 2 shows the structural relationship of the three components and the data flow information among them. The solid arrows indicate the control flow among the components, while the dashed arrows and dotted arrows show the input and output data flow among them, respectively. We shall first explain the data sources and then describe the functions and implementation scheme of each component.

There are three main input sources for F-menugen: menu configuration file, application property file and the session object. The menu contents and access control rules are specified in the menu configuration file, as described earlier. The application property file keeps the parameters specific to an application; it is the source for creating the *param* object. We provide a simple way for security administrators to specify such parameters: Java property files. Each application has its own application property file; F-menugen will process it and create a JavaScript object out of it. For instance, working days, office hours, and machines with special privileges can be expressed by the following name-value pairs in a designated property file.

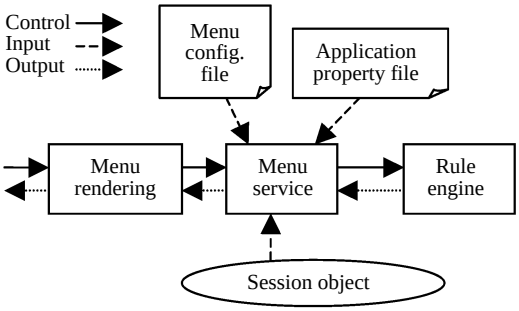

Fig. 2 The structure of F-menugen

```
weekday = Mon, Tue, Wed, Thurs, Fri
officeHours = 8,9,10,11,12,13,14,15,16,17,18
privilegedMachineIP = 1.1.2.1, 1.1.2.3
```

The session object is an essential component of any Web application. It keeps data specific to a user session, including user profiles such as name and title deposited by the authentication module. F-menugen will get all access control related user data from the session object and then construct a JavaScript user object out of them, one for each user. We note in passing that there may be concerns that when user data or access rules must be changed during a user session, the user object and thus the generated function menu will become obsolete. This can be addressed by installing a timeout servlet and turning on the auto refresh mechanism of the browser. Then when the browser refreshes its contents at specified intervals, the servlet filter will be triggered to see if is necessary to terminate the user session immediately, which in turn will re-start the authentication and menu generation process to generate an updated menu.

The menu rendering component is the "main" program of F-menugen. It is a JSP program (*menuRender.jsp*) called by the controller after the authentication module of a Web application has verified a user's identity successfully. When invoked, *menuRender.jsp* does some routine work and then asks the menu service component to generate the menu for the user. After

obtaining the menu as an XML string, *menuRender.jsp* will call a Java rendering method to transform it into an HTML frame to be displayed in the user browser.

The menu service component is the heart of F-menugen. It consists of several Java classes that are responsible for generating the menu contents in XML based on the access control rules and other inputs. There are four major tasks that the menu service component will perform. First, it needs to create a user object from data kept in the session object. Second, it processes the XML menu configuration file and extracts needed information using the schema-derived classes generated by Sun's Java Architecture for XML Binding (JAXB) tool (JAXB, 2003). This makes it very easy to process the XML contents. Third, it will transform the application property file into a JavaScript property object and make it part of the constraint evaluation context. Fourth, it traverses the whole menu tree to generate the tailored menu in XML for the current user. Along the way it will ask the rule engine to evaluate any access control constraints associated with the menu nodes specified in the menu configuration file. Note that, as the menu configurations and application parameters are both specific to an application system, the menu service component will cache their contents to avoid performance loss.

The rule engine component evaluates a JavaScript expression using the context provided by the menu service component. It is built on top of Rhino (Mozilla, 2002), an open source implementation of JavaScript in Java. Rhino was designed to be embedded into a Java application and it provides a full range of Java API for evaluating JavaScript code. Furthermore, it also supports defining JavaScript objects and functions in Java. Hence we can define additional Java functions and objects to be used in the access control constraints written in JavaScript. This is how we define the "*contains*" function and "*time*" objects described above. At runtime, the rule engine is instantiated as a singleton object and provides an "*evaluate* (*constraint, evaluationContext*)" method for determining if an access control constraint is satisfied for a user.

V. CONCLUSIONS

Security is attracting more and more concern in the development of Web applications. However, most current approaches to application-level access control focus on conducting application or data tier checks. In this paper, we have argued the need to develop a presentation tier access control support for Web applications, and presented our work towards this goal by presenting a flexible, accessibility-based function menu generator. Our access control rules specify which users can use which parts of the application's functions according to user profiles, application-specific parameters, and certain contextual information. A security administrator can specify various access control requirements, user grouping, and time-location constrained accesses, without actual coding. Furthermore, since all these specifications are documented in an XML configuration file, it is very easy to modify them and no source recompilation is needed. In the future, we plan to further simplify the task by developing an administrative tool with graphical user interface support for creating the menu configuration file.

REFERENCES

Apache, 2001, *The Apache Struts Web Application Framework*, Available: http://jakarta.apache.org/struts/

Beznosov, K., and Deng, Y., 2002, "Engineering Application-level Access Control in Distributed Systems," *Handbook of Software Engineering and Knowledge Engineering*, Vol. 1, S. K. Chang, ed.: World Scientific Publishing, River Edge, NJ, USA.

Curphey, M., Endler, D., Hau, W., Taylor, S., Smith, T., Russell, A., McKenna, G., Parke, R., and McLaughlin, K., 2002, "A Guide to Building Secure Web Applications," Version 1.1.1, *Open Web Application Security Project*, Sept. 2002.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1995, *Design Patterns*, Addison-Wesley, Reading, MA, USA.

Goodwin, R., Goh, S.F., and Wu, F.Y., 2002, "Instance-level access control for business-to-business electronic commerce," *IBM Systems Journal*, Vol. 41, No. 2, pp.303-317.

JAXB, 2003, *Java Architecture for XML Binding (JAXB)*, Available at: http://java.sun.com/xml/jaxb/index.jsp

JAAS, 2002, *Java Authentication and Authorization Service (JAAS)*, Available at: http://java.sun.com/products/jaas/

JSP, 2004, JavaServer Pages Technology 2.0, Available : http://java.sun.com/products/jsp/

Kouadri Mostefaoui, G. and Brezillon, P., 2003, "A generic framework for context-based distributed authorizations," In: *Modeling and Using Context (CONTEXT-03)*, Lecture Notes in Artifical Intelligence, Vol. 2680, Springer Verlag, pp. 204-217.

Microsoft, 2003, *ASP.NET*, Available at: http://msdn.microsoft.com/asp.net/

Mozilla, 2000, *JavaScript 1.5*, Available at: http://www.mozilla.org/js/

Mozilla, 2002, *Rhino: JavaScript for Java*, Available at: http://www.mozilla.org/rhino/

Manuscript Received: Sep. 12, 2005
Revision Received: Aug. 16, 2006
and Accepted: Sep. 03, 2006