Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Field-Oriented Approach to Web Form Validation for Database-Isolated Rule

Liang, Zhou
Institute of Computer Science and Technology
Zhejiang University
Hangzhou, China
zhouliang@zju.edu.cn

Jianling, Sun
Institute of Computer Science and Technology
Zhejiang University
Hangzhou, China
sunjl@zju.edu.cn

***Abstract* - We address why, and especially how, to represent Field-Oriented XML-based rule file in form validation for web applications. Motivated by the decades-old problem of frequently changing requirements for software development, we propose a descriptive solution to validation on web form data. By extracting all Database-Isolated Rules, which are to be implemented on both client-side and server-side, into standalone rule file in a Field-Oriented format, management of business rules on either side can be simplified to a great extent. The main advantage of our approach is a consistency guarantee of validation logic performed by presentation and application layer, as well as prominent clarity in representation and notable flexibility in modification. It's a tailored replacement for traditional realization of hard-code, which results in redundant work, tight coupling structure, and highly possible inconformity of rules between both sides.**

***Keywords* — Field-Oriented, XML-Based, Database-Isolated Rule (DIR), Massive-Form, Web Application**

I. INTRODUCTION

As framework and technology become more and more sophisticated, the importance of web application grows significantly in e-commerce and financial industry. Banks, investment companies and trading corporations have now provided a variety of software applications, no matter internal management or commercial use. Meanwhile, web forms become really large-scale and complicated, having numerous fields in a single form as well as tanglesome relationships between each other. Such forms are referred to as "Massive-Form" in this paper, reflecting its scale in size and complexity.

In turn, the requirement on efficiency and security becomes such a high standard. Among all aspects, data validation is a core function of any successful application, as it helps to make certain the data from user conforms to specific business rules. Traditionally, validation rules are implemented both on client-side and server-side by hard code. Script ensures efficiency by validating user input in browser, and giving errors instantly without accessing server; while server-side validation guarantees security for the entire application by executing complete business rules including checking against database.

However, hard-coding complex rules into both client-side script and server-side module becomes a disastrous task when business rules are always changing in order to take advantage of a potential opportunity or respond to a potential threat. Reusability of such implementation is terribly low, because all validation rules are buried in messy and scattered code. Worse still, presentation layer and application layer are usually developed by different teams with diverse designs, so there's a great possibility to introduce inconsistency between each side, especially after decades' modification and upgrading.

Obviously, development and maintenance of validation on Massive-Form is exceedingly troublesome and error-prone. Though great efforts have been made to find an easier solution for developers, such as Simfatic Forms [1], JQuery Validate Plug-in [2], Microsoft Access and Java/XML-based data validation Framework [3], they only focus on one side of client or server. Apache Struts supports validation on both sides with an XML-Based rule file, but together with all the frameworks mentioned above, neither of them has ever tried to support "Inter-Field Rule" validation, which refers to those rules in which the status of target field is determined by other fields in the same form, e.g. in a typical order placing process of Gold Futures transaction, "if Order Type is selected as 'Two-way', then both Profit-Making Price and Stop-Loss Price are mandatory". Meanwhile, rule engine products like JBoss Drools [4] and ILOG BRMS [5] are too complicated and bulky for web form validation. Compared with the rules processed by these engines, validation rule is static, simple and unordered. Once again, they just concentrate on server-side improvement, incapable of managing rules running on client-side.

To give an efficient, secure, and flexible approach to web form validation, what we propose is to store validation rules into standalone files, then use our framework to run these rules automatically, both for client-side and server-side. Our approach decouples Database-Isolated validation logic from hard-code application, ensures consistency between each side, and frees developer from tedious and labor-intensive work.

Our concept consists of two novel aspects. First, we introduce a fresh fundamental knowledge of representation formalism: Field-Oriented. Inspired by the idea of Object-Oriented, we treat each target field as an "Object". All related rules operating on this field are placed together under the same block. In plain terms, "One Rule Per Field". Second, we offer a set of XML tags for validation rule, which is specially designed to share rule file between client-side and server-side, as well as amongst heterogeneous platforms and operating systems.

II. VALIDATION ON CLIENT AND SERVER SIDE

For most web applications, both presentation layer and application layer have complex code implementing business

logic, of which validation rules accounts for a great portion. With various programming languages and techniques, form data validation is executed separately on both client-side and server-side, say, Javascript for front-end and Java for back-end.

Modern database also supports table or column validation upon insert/update/delete by "Check Constraint" [6], but most web applications still choose to implement entire validation on application layer for flexibility reason. Besides, database implementation not only postpones validation process, but also increases security risk by manipulating database directly when adapting to frequently changing requirements. Thus, this uncommon realization is not in the scope of our paper.

A. Client-side Script

Web browser downloads scripts from server and run these interpreted language code according to user's input and event. Compared with back-end validation, front-end check allows application to give error as early as possible, which improves user experience significantly. Script offers great efficiency by reducing the number of round trips to the server, together with server's stress on processing and responding to invalid requirements. Whereas, relying on scripts for client-side validation is unmanageable, inflexible, and non-portable.

- Browser configuration is not under control of web applications. User can stop using Javascript at will.
- Different browsers have different APIs. Adaptation to various environments takes much risk as well as tedious labor work.
- Script can't perform Database-Related Rule, which means for those validations that need check user input against database, client-side script is of no avail.
- Business rules are mostly hard coded, which directly leads to the tragedy of maintenance when requirements are fast changing. Developers would be drowned in messy relationships of all fields, and troublesome orders of function calls.

B. Server-side Validation Module

A commonly practiced way of server-side validation is to execute complete business rules in a validation-specific module in the application layer of the classic three-tier architecture. Programming language like Java provides supreme flexibility to organize complex rules. However, this type of realization is still too bulky and labor-intensive for modern web applications.

- When validating a Massive-Form with more than 20 fields, developers have to code thousands of lines to check each field separately, which makes the whole application full of repetitive and lengthy codes.
- When some business rules changed, the whole life cycle of software development (code, compile, debug, test, deploy) has to start all over again, which increases development and maintenance cost drastically.

We can conclude from above, for web form validation, front-end check is sufficient, while back-end check is necessary. They cooperate with each other, providing efficiency and security for the entire application.

However, we can also observe, for web applications that contain numerous Massive-Forms, how terrible the traditional implementation could be. As we can see from above, no matter on client-side or server-side, hard coded realization has to be updated frequently so as to keep up with the fast changing requirements. To make matters worse, locating related code that needs to be modified is quite a meticulous task for large system. Because the validation rules are coded on client-side and server-side separately, new logic has to be implemented on both sides, which causes unnecessary work and introduces risk of possible inconsistency.

Most likely, validation code uses event-action functions on client-side, which brings unwanted coupling and ordering concern. Even worse, maintenance work is normally performed by different people on each side, who have different philosophy and practice on software development. Original design could be easily destroyed after ages' change. Consequently, as function of application evolves, structure of code degenerates.

Finally, we reach the question "Why don't we end up hard-code era by storing validation rules that are used on both sides into a standalone rule file?" The next section would explain what kind of rule can be extracted from validation logic.

III. BUSINESS RULE FOR VALIDATION

According to the Business Rules Group [7], "a business rule is a statement that defines and constraints some business. It is intended to assert business structure or to control or influence the behavior of the business. The business rules which concern the project are atomic, that is, cannot be broken down further."

Nevertheless, business rules that we concern in this paper are much more simple and specific. Business rules that are used on both client-side and server-side for web form validation doesn't require too much runtime execution, nor sequence management [8], that's because all fields should have already finished inputting when a form is being submitted. So conceptually, validation rules are static, explicit and unordered.

We can divide validation-specific business rules into two categories on the basis of their relationships with database.

A. Database-Isolated Rule (DIR)

In this context, "Isolated" means "Unrelated". As the name implies, this kind of rule doesn't require data from database, which means rule execution could be done without round trips between browser and server. Such rules could and should be implemented on both sides of a web application. An example of this category is "If A is selected, then B is optional".

B. Database-Related Rule (DRR)

This kind of rule requires database access in order to verify user input against data stored in database, such as check the validity of an input password for a certain user when logging on. DRR can be implemented on server-side alone, because script language is designed to run on client-side, which determines its intrinsic inability to access database directly. The only way to get data for script is to make requests to server and let application layer play the role of data carrier.

This paper would only pay attentions to the first category. Running a DRR on client-side is wasteful and meaningless,

since passing database data to client-side validation loses efficiency, which violates the original purpose of script.

Technologies like Ajax [9] raise a brand new idea on creating interactive and rich web applications. With Ajax, applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. Though the use of asynchronous requests allows browser to respond quickly to user inputs, and sections of pages can also be reloaded individually, it's not quite necessary to use such technology in non-interactive form validation. The time spent in request/response may not save any time comparing to running these rules just at server-side.

IV. DATABASE- ISOLATED RULE

To be more clarifying, we would classify the Data-Isolated Rule (DIR) into two subdivisions, according to the number of fields that are connected in one rule.

A. Single-Field Rule (SFR)

This subdivision of DIR only verifies one single field, independent from other fields in the same form. Validation logic in SFR normally appears very simple, such as checks on length, format, or number range. Numerous products [1, 2, 4] have covered SFR validation work and achieved great success in industry. SFR typically provide the first layer of validation. When designing a web form, business analysis team defines SFR for each field, which restricts what users can enter.

B. Inter-Field Rule (IFR)

On the opposite of SFR, an IFR connects more than one field together within itself, which means at least two fields in the same web form are related in this rule, one being the validation target, and the rest being condition suppliers. The logic for IFR could be really complex and flexible. As a result, most applications use hard code realization to implement IFRs. Though rule engines are invented and introduced to help out the cumbersome work, they only work for server-side.

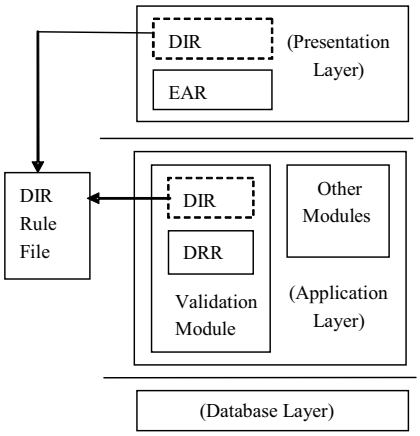

Figure 1. DIR file in 3-tier web application

Our goal is to extract both SFRs and IFRs from business logic into a standalone rule file, decoupling DIR from hard coded application. With our framework, developers on presentation layer can concentrate on Event-Action Rule (EAR) like "onChange" function, while those who work on application layer could emphasize on DRR and other dynamic process rules. For DIR, we take over the translation and validation work automatically, which not only lowers cost for development and maintenance, but also ensures consistency on both sides and increases reusability of rule component.

For security and efficiency reason, our framework can even converts the rule file into a corresponding Javascript file in advance, when deploying, developer can simply put this pre-generated file into the correct folder and use it afterwards.

As we can see from Figure 1, after DIR become external to the web application that depends upon them, the variable validation logic can be easily updated by changing the rule file alone, without intruding the structure on both sides.

V. FIELD-ORIENTED AND XML-BASED

No matter in what kind of programming language, branch syntax like IF/THEN/ELSE shows great power and flexibility in dealing with conditional situations, making itself one of the most essential element in any programming language.

But from the standpoint of a web application developer, branch syntax is also the root reason of all risks and problems. Too many IF/THEN/ELSE would degrade readability, reusability and flexibility sharply.

To be more illustrative, we propose a pair of conceptual terms about branch syntax, which would be further used.

- *Target Field:* the status of it is to be checked, always in "THEN" block, being target of a validation.
- *Condition Field:* the status of it serves as an evaluation term, mostly in "IF" block, being condition supplier.

JBoss Drools [5] gives similar concept when introducing <LHS> and <RHS>. Each item of validation rules in "THEN" block contains the constraints that should be satisfied for a target field when the "IF" evaluation returns "true". This is an inherent mapping from business rule, because that's how a rule is defined in specification documents.

However, such realization brings big trouble when each field of Massive-Form has complex validation logic, that is, the status of target is determined by a complex combination of condition fields' status. Under this circumstance, validation code for this target would be scattered throughout the entire application. When business rules change, as they often do, the maintenance becomes terrible and tragic. Rediscovery of related code is quite painful, while the entire process of software engineering is even labor-intensive and error-prone.

To overcome the drawback of traditional rule assignment by hard code, considering the traits of form validation (static, explicit, unordered, non-interactive), we take a fresh break in extracting DIR from business logic by representing them in a Field-Oriented way in standalone rule files.

"One Rule Per Field"—a common parlance to describe the idea of "Field-Oriented". In our approach, all the DIRs on a target field are assembled under the same Field Rule tag, which can be considered as an exclusive block of validation logic of the target field. By reversing conventional process of coding, the ruled becomes the ruler, the passive turns into the active.

XML is an ideal technology to represent and store DIRs. With its portability, interoperability and extensibility [10], the extracted DIR file can be easily shared, managed and revised.

For client-side, all modern browsers have a built-in XML parser that can be used to read and manipulate XML. Parsers read XML file into memory and converts it into an XML DOM object that can be accessed and further operated by JavaScript.

For server-side, there are more choices. Though DOM4J [11] is considered the most powerful library to process XML, W3C DOM [12] is used in our framework in order to keep openness to all programming languages rather than Java alone.

With the help of XSL, XPath and XQuery [13], we can display the rule file in a clearer and user specific way. This can make our implementation more understandable and readable for business team [14]. User can even generate rule report on certain fields or conditions.

So, finally, our DIR file would be expressed in an XML-Based and Field-Oriented format.

VI. DESIGN

Before introducing detail structure, first let's see how many kinds of DIR are commonly used in Massive-Form validation.

a) Format
b) Number Range
c) Legal Value Set
d) Calculation Check
e) Coexistence Check
f) Mandatory Check

The first three are always SFR, since they only perform constraints on a single target field. Thus, as always does, they are represented as XML attributes for each target field.

The following two are obviously IFR. The rule itself determines at least two of fields are bound together in processing validation logic. Calculation Check validates the value of numeric target against a result of calculation on other fields under certain conditions. Coexistence Check constrains the value of target within a specific list according to the combination of status of condition fields. These two types of DIR are represented as sub children to the Field Rule.

The last is somehow a fence-sitter, being either an SFR or an IFR. We have situations in which one field is always mandatory while in other cases conditional mandatory, former SFR and latter IFR correspondingly. Taking a further thought, we find the conditional mandatory presence check is a special case of Coexistence, by allowing any value. Considering the simplicity and usability, we represent Mandatory Check Rule as an attribute for SRF and as Coexistence Check for IFR.

A. Root Element

The root element of the XML rule file is <formValidator>, with attributes "id" and "include".

For server-side, the rule file is parsed and executed at run-time. For client-side, we can process it in browser. Yet, taking security and efficiency into consideration, our framework can also translate the rule file into Javascript file first, with the same name as the "id" attribute. Then server pages like JSP only need to include a reference to this pre-generated Javascript file in <script> tag.

As all programming language does, we also give the ability to use external files, mostly, containing definitions of all Function Rules. The "include" attribute locates these files.

B. Field Rule

Each Field Rule serves for only one target field in the form exclusively, embodying the concept of Field-Oriented. All the DIRs on the target field are registered under <rule> tag.

By convention, all SFRs are represented as attributes of Field Rule. Defined attributes include "id", "target", "minlen", "maxlen", "format", "range", "required", "legalVal" etc. Validations on SIF have been tried out by many frameworks [1, 2, 3, 4], both for client-side and server-side, so we don't expand these topics here. The only one we'd like to emphasize is the "target" attribute, the value of which should be the same as the id of this field in HTML code.

C. Coexistence Check

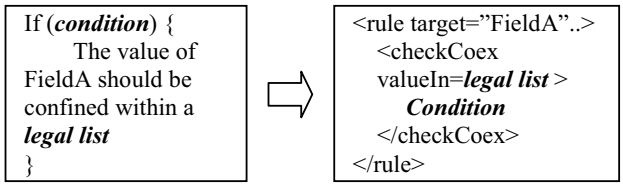

Figure 2. Structure of <checkCoex> tag

There are situations under which the value of target field is dependant on a combination result of condition fields, and the value should be confined within a specific list of legal value set, mostly for a drop-down.

<checkCoex> is designed to execute such validation. All the conditions are put together under this tag, with an attribute of "valueIn", holding the legal value set as a string by separating each value with comma.

As we mentioned previously, conditional Mandatory Check is a special case of Coexistence Check. By assigning "valueIn" with "*" (representing any string), the coexistence rule turns into an instance of conditional mandatory check rule.

D. Calculation Check

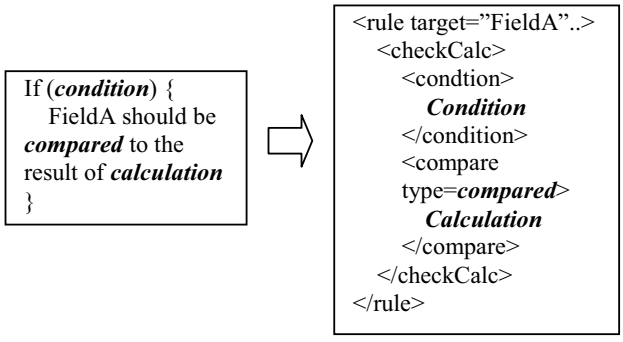

Figure 3. Structure of <checkCalc> tag

In considerable cases, for a numeric target field, the value of it should be compared (equal, less, or greater) with a result of complex calculations on several other fields. <checkCalc> is used to deal with such kind of rules.

Unlike <checkCoex> who is a born condition check rule, <checkCalc> concerns on the result of calculation comparison under certain conditions, so we divide it into two sub blocks, <condition> and <compare> respectively.

<condition> holds the combination of conditions, while <compare> focuses on the result of calculation. The "type" attribute of <compare> indicates what kind of comparison should be executed, "eq", "gt" or "lt", inside which, calculation detail is described with various mathematical tags. The result of this calculation is then verified against the value of target field.

E. Error Message

Giving correct error message is always an important task for first-class products. Our framework supports system error as well as custom specific error. By instantiating <errMsg> at the end of a rule block, application would raise a customized alert when validation fails. Though these errors are most likely to be triggered in client-side, we also ensure this function on server-side by printing them in log files.

F. Function Rule

In business rule, it's common that under the same condition, more than one field is to be validated. Enlightened by syntax of Business Rules Markup Language [15], we introduced the concept of "Function Rule", which is a general-purpose functional rule that can be used in many situations.

<defFuncRule> indicates the code inside defines a Function Rule. Say, for business logic "if A1 and A2 are null, then B and C and D should be mandatory", we can define a Function Rule named FR_A that evaluates whether A1 and A2 are null, if true, then the invoker should not be empty, otherwise, an error message shows up.

Then, the corresponding Field Rule for field B, C and D calls FR_A in their own rule to test nonempty constraint respectively, by using <useFuncRule> with its "funcRuleId" attribute referring to "FR_A".

G. Parameter

To define and utilize Function Rule, behavior of parameter is a great concern for structure design. In <defFuncRule>, attribute "paraNum" indicates the number of parameters this function takes, while another attribute "para" can be used alternatively to expatiate each parameter's type, offering validation on data type. When getting called, "para" attribute of <useFuncRule> is used to send all parameters to the callee.

We use "@para" followed by an index to identify the corresponding parameter that is passed in <useFuncRule>. E.g. when declaring <defFuncRule>, user can take "@para0" to get the value of the first parameter, so on so forth.

The caller name is usually used in error message, as well as in calculation, so we introduce "@caller" to identify the invoker's id. A configuration file mapping this id to its screen name is set up in the system. So the @caller would be finally replaced by the actual field name when displaying the message.

By providing parameters, our framework earns more flexibility and simplicity. Commonly used function rules can be extracted and shared between target fields.

VII. EXAMPLE RULE FILE

The code segment shown in Figure 4 demonstrates how the DIRs on "Profit Making Price" (PMP) field are represented in a Field-Oriented and XML-Based format, in a typical pending order placement of an online gold trading application.

```
<formValidator id="FV_OrderPlace">
  <defFuncRule id="FR_ProfitMaking" paraNum="0">
    <checkCoex value="*">
      <field id="F_OrderType">ProfitMaking</field>
      <errMsg text="@caller should be mandatory when
      Order Type is Profit Making!"/>
    </checkCoex>
  </defFuncRule>
  <rule target="F_ProfitMakingPrice" maxlen="10"
  type="number">
    <useFuncRule funcRuleId="FR_ProfitMaking" />
    <checkCalc>
      <condition>
        <AND>
          <field id="F_OrderType">ProfitMaking
          </field>
          <field id="F_BuyOrSell">Buy</field>
        </AND>
      </condition>
      <compare type="gt">
        <field id="F_CurrentBuyPrice"/>
      </compare>
      <errMsg text="Profit Making Price should be
      greater than Current Buy Price!"/>
    </checkCalc>
  </rule>
…
</formValidator>
```

Figure 4. Example of DIR file

This last example of validation rule file can be described in readable language as:

1. PMP is a number type field, with max length of 10.
2. PMP is mandatory when "Order Type" field is "Profit Making".
3. When "Order Type" field is "Profit Making" and "Buy or Sell" field is "Buy", the PMP value should be greater than the value of "Currrent Buy Price" (CBP) field.

Figure 5. Readable transalation of example

In the <defFuncRule> block, a function rule named "FR_ProfitMaking" is defined, taking none parameters from caller. If the value of "Order Type" field equals to "ProfitMaking" but the caller is null or empty, then an error message is raised, in which, the "@caller" specifies the invoker's id. This function rule can be used by any other Field-Oriented rule whose target is also supposed to be mandatory when "OrderType" is "ProfitMaking".

Then, this defined function rule is called in the exclusive Field Rule of PMP, by using <useFuncRule> tag with "funcRuleId" pointing to "FR_ProfitMaking". Since there's no parameters needed, the attribute of "para" can be omitted here. Another inline DIR is declared after <useFuncRule>. This <checkCalc> rule is used to compare the value of PMP against CBP, ensuring the former to be greater than the latter. If not, an error is alerted to inform user to correct their input.

For conciseness, the example we choose doesn't have very complex logic, but still, it shows great adaptivity and simplicity in representing DIR.

When it comes to real financial system or investment application, the forms would become really large-scale, normally more than thirty fields with complicated relationships between each other. The Field-Oriented XML-Based approach offers an opportunity to represent DIR in a clear and simple way rather than hard coding everything into application which messes up business logic as well as complicates development and maintenance work drastically.

VIII. Discussion

Though the process to extract rules from existing code into human readable language is not the focus of our paper, lack of rule extraction tool [16] brings limitation to the usage of our framework. Our solution is more suitable for an application that is well documented or one that is just to be started from scratch than one with little documents or a legacy system.

Although most of current web applications use XML to store data and communicate between modules, there's still a slight possibility that some systems use other format as intermediary. Our framework can't be applied to such system, since XML is an inherent requirement of our concept.

We offer type check under some circumstances, however, the concept of data type is not being attached too much importance in our solution. Because we suppose form validation doesn't require too many checks on parameter's type.

The framework realizing our approach is still a prototype. Current version supports Javascript for client-side and Java for server-side, with limited functional tags. But for sure, supports on more languages and tags would be implemented in future. Apart from horizontal extension, more auxiliary functions would be augmented in following release. Rule report on user specific criteria would be quite helpful for communication between business team and develop team.

Researches on extracting and unifying validation rules for classic three-tier (presentation/application/database) system would be expanded based on the concept of this paper hereafter.

IX. Conclusion

In this paper, we outline a representational solution to form validation for web application. First we introduce the client-side and server-side validation techniques that are broadly used in industrial and commercial area at present. After listing the major drawbacks of conventional implementation on both sides, we then bring forward our approach of representing validation rules in standalone rule files which can be parsed and used by both sides during run-time. As a positive result, web applications can minimize the impact of fast-changing requirements on software development and maintenance. Categorization of validation rules is made according to their relationships with database, which are DIR and DRR respectively. A further subdivision is specified within DIR, on the basis of the relationship between target field and condition field in one rule, namely, SFR and IFR.

Inspired by Object-Oriented concept, we introduce the idea of "Field-Oriented". The meticulous and burdensome searching through scattered code to update validation rule can be simplified to a large extent by applying this novel concept. Meanwhile, XML is considered as a perfect format to store DIRs, which are static, explicit and unordered in essence. Externalized standalone XML-based rule file is well structured and clearly represented. By gathering all related DIRs together under the Field Rule tag of target field, the implementation of "One Rule Per Field" gains more flexibility, consistency and reusability to web form validation, as well as more fine-grained control of data validation with comparison of conventional hard-code realization.

Various general purpose tags and attributes are utilized to depict DIR in a concise and understandable format. These tags and attributes can be parsed and verified automatically within our framework, which easily accelerate development process.

Thereby, our new concept supports easy management of validation rules, guarantees consistency between client-side and server-side, as well as decouples DIR implementation from both sides. With the help of our framework, form validation becomes much more convenient and labor-saving, offering reliable and stable maintenance to the entire web application.

References

[1] Simfatic Forms, http://www.simfatic.com/, 2009

[2] JQuery Validate Plug-in, http://docs.jquery.com/Plugins/Validation/

[3] J. Xue, "Java and XML: Learn a Better Approach To Data Validation", http://www.devx.com/Java/Article/16407/1763, 2002

[4] JBoss Drools, http://www.jboss.com/products/rules/, 2009

[5] ILOG BRMS, http://www.ilog.com/products/businessrules/, 2009

[6] Database Check Constraint, "Sams.MySQL, 3rd.Edition"

[7] D. Hay and K. A. Healy, "Defining business rules - what are they really?", http://www.businessrulesgroup.org, 2009

[8] F. Rosenberg and S. Dustdar, "Business Rules Integration in BPEL – A Service-Oriented Approach", In Proceedings of the 7th International IEEE Conference on E-Commerce Technology (CEC'05), 2005.

[9] Ajax, http://www.ajax.org/, 2009

[10] B. McLaughlin, "Java & XML, 2nd Edition", pp. 10-17

[11] DOM4J, http://www.dom4j.org/, 2009

[12] W3C DOM, http://www.w3.org/DOM/, 2009

[13] E. T. Ray, "Learning XML, 2nd Edition"

[14] M. Teraguchi, I. Yoshida, and N. Uramoto, "Rule-based XML Mediation for Data Validation and Privacy Anonymization", IEEE International Conference on Services Computing, 2008

[15] B. N. Grosof and Y. Labrou, "An Approach to using XML and a Rule-based Content Language with an Agent Communication Language", In Proc. IJCAI-99 Workshop on Agent Communication Languages, 1999

[16] V. Wadhwa, "Method And System Of Business Rule Extraction From Existing Applications For Integration Into New Applications", Patent No.: US 6389588 B1