Business Rules in User Interfaces

by Kamlesh Pandey

The business rules approach has revolutionized information systems, making them more agile and adaptable to changes in business logic. However, the adoption of this approach has been asymmetric across different layers of information systems. One of the areas that has remained largely untouched is the user interface. This paper looks at the need, the benefits, and the technological feasibility of using the business rules approach in user interfaces.

A Glance at the World of Paper

Before we look into the paperless world of information, let us see how things work in the paper-pushing enterprise. A paper form is a static user interface that allows creation of data upon its processing. There are constraints applicable to this data, usually present as instructions for filling out the form. The user can still fill data that violates these constraints. These violations are supposed to be trapped during the processing of data upon submission. The user can be informed about the errors as a separate process.

A similar process works for the presentation of data. When your bank sends you an account statement, it is presenting you a static, read-only user interface that is governed by constraints. For example, the interest on deposits must have been calculated using the rules governing interest rate payments for that account type.
User-friendly Interfaces

As we march towards a paperless (and perhaps greener) world, there is substantial change in the way a user sees a user interface. The user no longer goes through the instructions for filling the form but expects to be guided through the form-filling process. There are many things that distinguish a 'user friendly' interface from a clumsy interface. In this section we would look few of these where business rules can play a role.

1. Relevant Information

Anyone who has suffered the misery of filling out a long paper form knows the ubiquitous "Not Applicable" fields. Sometimes the instructions require us to cross them out, or still worse, write "Not Applicable." A well-designed user interface should display only relevant information and data collection fields.

2. Instant Feedback on Errors

Each piece of information being captured on a form is interrelated to data collected elsewhere on that form or to information in the database. Each change in the data has the potential for violating the constraints. A user-friendly interface should give instant feedback on such errors. The user should not be forced into an endless correction cycle where the first submission results in few errors, and when the user corrects these errors a new set of errors is reported back.

3. Auto Initialization

In the world of paper, a completely blank form is a common sight. However, in the case of electronic forms there is a huge scope for auto initialization of fields using rules. For example, if the user has entered the city in an address, the user interface can initialize the country field with the country from a geographic database.

4. Aid in Calculation

Compared to the paper forms that force the user to do all the calculations, their electronic counterparts can help the user in calculations. For example, given the line item details a form can display the total order value.
5. **Visual Clues**

To make the data entry faster and to achieve a better user experience, the form can guide the user in data entry. In India, for example, any financial transaction above 50000 Rupees requires the user to provide a tax ID. A well-designed user interface should have an optional tax ID field. If the user fills a transaction amount that is greater than 50000, the user interface should give a visual hint to the user (e.g., a red asterisk) that the tax ID field is mandatory in this case.

6. **Authorized access**

In the world of paper, we have the "For Official Use Only" sections that a user is not supposed to fill. Conversely, when a form is processed, the processing agent is not supposed to change the data entered by the user. This problem can be generalized into a statement that each user must be able to change only those fields that he/she is authorized to change. Sometimes this authorization can be dynamic and a user may or may not be allowed to change a field depending on what data has been entered up to that point. A well-designed user interface should make the fields editable only if the user has read/write access. Furthermore, if the user doesn't have access to view the information, the data should not be visible.

From the user's perspective, all these expectations are quite genuine. In our daily lives we deal with a number of user interfaces for online ordering, registrations, and so forth. With the technologies of the day, it should not be difficult to meet these expectations. However, it too often comes as a surprise that a significant number of user interfaces fail to meet these expectations.

**Thick-skinned Thin Clients**

User interface design was revolutionized by the use of internet browsers and markup languages for the presentation and capture of data. The business logic was forced back into the server side. While this approach works quite well, there are many shortcomings of markup languages. An HTML page is a mix of presentation markup with untyped and unstructured data. To provide any intelligence to the user interface, we have to rely on client-side scripting or communication back and forth with the server.

Even now, it is not unusual to be forced to fill in the city, state, zip code, and country,
when it is possible to auto initialize the other fields using zip code or to present country/state (choices) when given a city. A message informing that certain fields are required, only after submission, seems to be a norm. Even more frustrating is the experience of filling a multi-page loan application form that keeps shuttling back and forth between a dumb client and a slow server before finally getting to a "your request has been submitted" outcome. If you think I'm exaggerating, I haven't yet mentioned those annoying script errors or data loss due to server errors.

The technological advances of the past few years have increased the processing capabilities of browsers. The emergence of XML-based technologies has paved the way for creating smarter user interfaces, which work with structured and strongly-typed data, governed by business rules, while retaining all the benefits of thin clients with standardized browser-based user interfaces.

**Business Rules in User Interfaces**

Anyone new to the business rules approach might think that user interface would be the last place to look for rules. This is not true. The user interface is the gateway to the real world; it provides the boundary where a system interacts with the end user. All the constraints applicable to the information in the real world are applicable here. If you look at the BRS Rule Classification Scheme by Ronald Ross[1], you will find that user interfaces contain all the types of rules mentioned there. Here are a few examples.

1. **Presentation Rules**

   These rules can change the display properties of the data fields. Using these rules you can 'show' or 'hide' a field. You can make a field mandatory or optional ... editable or read only.

   Examples:

   - If there were accidents last year then the accident details field must be displayed.
   - If the accident type is fatal the accident description field must be mandatory.
   - If the ticket type is 'One way' then the return date must be made non-editable.

2. **Data Validation rules**

   These rules enforce a constraint on the data.

   Example: A single order cannot contain items from more than one category.
3. **Calculation/Inference rules**

These rules calculate or infer values based on other data values.

Examples:

- Shipping cost is calculated as $5 for shipping within the U.S. and $10 for shipping outside the U.S.
- If the credit score is less than 500 the customer is a high-risk customer

4. **Initialization rules**

These rules initialize the value of a field from other fields. The key difference from calculation rules is that the initialized value may be changed by the user.

Example: Initialize the billing address with the shipping address.

5. **Executive rules**

These rules execute a procedure, usually a client-side script, upon certain events.

Example: When the user clicks 'calculate' then calculate the EMI.

Apart from the simple examples given above, there are certain types of user interfaces that absolutely need business rules. For example, in workers' compensation insurance, there are more than four thousand class codes that determine the workers' compensation premium rate commensurate with the risk associated with that workplace exposure. Each class code requires a different type of information to be collected, and sometimes the information requirements for the same class code vary from state to state. In such cases the user interfaces are highly dynamic and business rule driven. The answer to one question can trigger further questions in related areas. Sometimes these user interfaces have to be built dynamically using a decision tree. These user interfaces can gain tremendous flexibility if a business rules-based approach is used.

**Declarative Representation, Automated Enforcement**

Having seen the role of the business rules in user interface, a key question arises. Should we execute the rules in the user interface? Can we not execute the rules at the server and keep the client free from the complications of rule execution?
The demands of user-friendly interfaces require us to make our user interfaces intelligent and responsive. This requires putting logic in the user interface layer. However, there the reliance on scripting languages to execute business logic suffers from the same problems that led us to use a business rules approach on the server side — to name a few of these problems: frequent changes in business logic, the need for visibility to the business rules, and the need to manage rules as an independent artifact. This brings us to two basic requirements of business rules-based systems.

First, the representation of the business logic must be imparted **declaratively** to the user interface. It should be possible to change the business rules without having to change other aspects of the user interface. Similarly, the changes to the presentation aspects should not affect the rules. The HTML-plus-scripting combination suffers a big disadvantage in that it has the data and the presentation mixed together. The business logic is written in scripts that must refer to the user interface components that extract data. Suppose you have an input field that allows you to type in the country. If you change this field to a dropdown list, the logic of accessing and setting the value of the country will change.

**Automated enforcement** of business rules by the rule engine is the second key aspect of a rules-based approach. However the script-based execution of business logic is typically tied to HTML events. Suppose you have two fields 'Start Date' and 'End Date' and you want to author a constraint that the end date must be greater than the start date. Suppose you write a script that gets triggered when the user enters an end date and checks the validity of the value governed by the constraint. Will this guarantee the validity of data? No, because the user can enter a valid 'End Date' and then go back to the 'Start Date' and change it to a date after the 'End Date'. In the scripting approach you will be required to ensure that the same script is executed when the 'Start Date' is changed. The constraint can still be violated when the page loads up with incorrect data.

A business rules-based approach would rely on a simple solution. All it takes is a rule "End date must be greater than Start Date" and a rule engine to enforce that at all times. However, that brings us to an implementation challenge. Is it possible to have a business rules engine running in a web browser?

**Emerging Technologies**

The good news is that the technologies to fulfill this vision have already come into existence — most notably, W3C's **XForms** specification.[2]
XForms is an XML application that represents the next generation of forms for the Web. By splitting traditional XHTML forms into three parts—XForms model, instance data, and user interface—it separates presentation from content, allows reuse, gives strong typing—reducing the number of round-trips to the server, as well as offering device independence and a reduced need for scripting.

In XForms, the user interface fields are bound to the data elements using binding expressions. The binding elements also contain property expressions that determine the values of various properties of the data. As depicted in Figure 1, the linkage between the data and its user interface field results in changes to the state of the field whenever the property values change.

Fig. 1. XForms linkage of data and user interface

XForms provides a declarative way of authoring the constraints and presentation rules. In the near future browsers will support XForms, and this will reduce the need to write the rules in a scripting language.

The syntax of expressions in XForms is a formal rule expression in a language such as XPath. However, the business rules management tools can allow the business user to write rules in natural language and then convert these rules into execution-friendly formal rule expressions. One example of this approach is Blaze Advisor's Smartforms, which allows designing forms based on a data model using drag and drop. It also allows users to write business rules and associate these with forms. The rules are
executed in the browser by a lightweight rule engine. By using XForms an underlying technology, this approach provides the benefits of a business rules approach while hiding the complexities from business users.

Write Once, Use Anywhere

Consider this. An enterprise provides a web-based interface for submitting a purchase order form. The enterprise also provides a web service that accepts purchase order forms in XML format. The submitted data is processed and stored in a database. Each subsystem responsible for creating, manipulating, or storing any part of the data must ensure the consistency of the data. Given the likelihood of heterogeneous environments for these subsystems, would it not be better if we could write the business rules only once and use them in the various execution environments? The answer is a resounding "Yes!"

To avoid redundancies and to ensure better maintenance of business rules, it is imperative that the business rules management products provide ways to apply the same rules in different execution environments. This may sound idealistic but even here the technologies of the day have opened up immense possibilities. For example, the representation of data in XForms is strongly typed and data validation takes care of enforcing all the constraints that are defined in the XML schema. Furthermore, the same set of business rules can be enforced in the browser (for data collection) and also be applied at the server side on data streams coming to web services.

Conclusion

Business rules can play a vital role in making the user interfaces adaptable and user friendly. The emerging technologies in the internet space are opening up doors for innovative solutions. The business rules vendors must come forward to tap this opportunity to build intelligent, dynamic, and user-friendly interfaces.

References


Kamlesh Pandey is a software engineering manager, leading the Blaze Advisor development team at Fair Isaac India. He is the architect of the Smartforms for Blaze Advisor, a product extension that allows designing rule driven user interfaces capable of executing business rules on browsers.

In the past 10 years, he has worked extensively on the development of decision support systems in the finance, E-Commerce, and petroleum industries. His work has been focused on creating rules-based frameworks utilizing enterprise architecture concepts and technologies. He can be reached at kamlesh.pandey@gmail.com.