Introduction

The authors are actively involved in the architecture and continued development of a rule management system that is targeted on simplifying complex specifications and technologies to end users. The developed solution is referred to as Semanak and implements familiar UI behavior while wrapping complex aspects of many interrelated technologies to present a simplified approach to developing, managing and deploying rule based business requirements. The learning experience derived through this real application development process that targets end users and application needs, appears to share overlapping commonality with your initiative. Below are some of the items and issues that we can bring to your efforts.

Shared Business Rules in XML

Easily configured sets of rules continue to be a mainstay for business intelligence. There exists a need to share these rules across various platform and organizational boundaries. As the usage of shared XML vocabularies increases (e.g. Address, Applicant) the need to combine and reuse these sets of rules grows. Leveraging XML and existing XML related recommendations simplifies implementation and reusability.

> *Semanak was developed to allow the reuse of business rules. Though it was initially designed for Mortgage Banking/Lending scenarios its usage extends beyond this arbitrary boundary. In its design it treats rule evaluation as a process that follows: (1) Collect inputs, (2) Process decision tree, and (3) Produce output.*

Process

Separating the steps of a rule evaluation into discreet areas allows you to enhance and extend each area separately from the others. In terms of specification this allows implementations the ability to focus on specific areas for conformance.

Input

Because there is a wide array of inputs that could be used in rule evaluation it is best to treat each value within a rule expression as a distinct entity. Often these entities can be stored within the same physical or logical location. Some examples of input sources include:

- Text Document
- XML Document
- WebService Value
- XPath Value
- SQL Query
- Parameter (e.g., HTTP, UserName)

- Programming object or script

Clearly this list is not meant to be exhaustive, but demonstrate the varied nature of input sources. It is our position that these inputs should be specified in a collection of "values" which can then be used within the rules themselves. This allows the implementation the ability to cache source information and update specific values as necessary throughout the rule process. Much like XPath, values should allow simple atomic data as well as vector/array items.

Maintaining the mappings of values to input sources in an extensible format such as XML is imperative for maintenance tasks and extensibility.

Processing

The intended process of evaluation can strongly influence the representation of rules and rule languages. When working with rules as conditional evaluations, a hierarchal representation is fundamental and processors tend to operate in a recursive mode. Additional approaches such as graph analysis (when working with labeled documents or topic maps) or utilization of inference engines (when working with semantically encoded data and an OWL document) are equally viable.

Though there are varied types of processing for rules, our current focus is on using conditional logic to process business rules. Ultimately this means that each rule operates on a set of values or constants and can be reduced to a Boolean equating to pass or fail. Within our existing implementation we operate on a hierarchal set of rule expressions. Each expression can contain multiple children which may be evaluated based on whether or not the parent expression evaluates to true or false.

It is possible to limit conditional rule processing to a functional paradigm. Within our initial implementations we allowed the possibility to export a set of rules (when only XPath values were used) to XSLT. Limiting an implementation to features available in XSLT was very useful for purposes of sharing rules, however specific limitations created problems. Ultimately with a specification specific to rule management such limitations can be avoided.

In addition to evaluation of rule expressions, it can be useful to allow additional operations in the process. Assigning new data to creating values allows end users the ability to create much more complex systems of rules. Labeling and jumping to specific sections of rule hierarchies can be equally useful. (Once specified, however it might prove to be more interoperable to instead <include> or <use> rule sections at specific points).

Outputs

Like inputs, the varied forms of output require that implementations bind to them in a separate step. Each output may or may not allow you to insert fixed text, a value (which may have been modified during process), or the result of an evaluation. Additionally, a collection of outputs may or may not be aggregated to produce any kind of output document (usually text). Again, this can work similar to result documents from within XSLT.

Binding to a Grammar

In addition to creation of "value" objects for each discreet piece of information from an input source, it might simplify an end-users job to bind a set of values to a specific schema by the names within the grammar. The schema may be represented as a DTD, an XML Schema or RELAXNG. In all three cases however it should be possible to bind the value to a specific namespace (or the null namespace). This allows the user to execute their set of rules against documents that contain items from a specific namespace, while ignoring unknown names. Built-in support for compound document processing should be considered high priority for the specification.

Activation and Expiration

While it is possible to filter input data based on namespaces, additional features are required to filter rule expressions, values, and input sources based on activation and expiration information. For example, a compliance rule in the mortgage industry might begin on January 1st. Instead of requiring the end-user to replace a rule-set document on January 1st at midnight it is preferable to have a document that contains two rules-- one which expires on January 1st and one that is activated on January 1st which supports the new compliance test.

Version Control

The ability to manage and control rules by some versioning scheme is required. In many cases an effective rule system will replace the need to compile business requirements and rules in code. And just as with code versioning, the rules should have built in versioning as well.

When Data is Not Available

There will be many cases where the data for a specific value is not available at the time of processing. There should be at least three options for handling this circumstance:

- Use a default value
- Stop the process
- Skip the value

Allowing the implementation to skip unavailable values is useful for allowing partial result sets. Additionally, it is possible that a specific value is not present because it will not be used. For example, consider an ApartmentNumber value. Clearly this value is not important if there is no ApartmentNumber and it should not stop the process.

In cases where a value if not available and it was marked as skipped additional information is needed when evaluating expressions. If a rule expression is encountered that uses a value which is not available and has no default, the processor must allow at least two options:

- Stop the process
- Skip the rule

Two additional possibilities include:

- Always pass
- Always fail

This allows the user more complete customization in exceptional circumstances.

Conclusion

XML and its associated recommendations can be used to simplify and standardize rule languages and the evaluation of rule languages. By dividing the process of rule evaluation into discreet steps implementers can specialize and extend the various sections. Ultimately this will enable varying input sources, processing techniques and output formats.

Jeff Rafter jeff@jeffrafter.com