Position Paper for OOPSLA 2002 Workshop:
Using Domain Specific Languages to Drive Business Applications

Design considerations for applications that use domain-specific languages

Jim Tyhurst, Ph.D.
Tyhurst Technology Group LLC

Contents

1. Introduction

Over the last two years, I have worked on several applications that I would characterize as "dynamically configurable" by domain experts. In one case, the application provides a general tool for capturing domain knowledge about structured events, such as radiological observations or specifications of pharmacological studies. Domain experts develop decision trees that guide other users to choose among the alternatives, so that data regarding specific events can be captured in a structured and consistent manner. In addition to representing domain expertise, the knowledge base has certain syntactic rules that enable it to be rendered in natural language.

In a second and very different application, marketing executives and medical professionals create business rules that are used to create dynamic questionnaires for medical patients. The tool is quite specific to producing a survey, although it is not limited to medical questionnaires. It could be used in any situation where a customer undergoes a prescreening process before a consultation with an expert, such as a student meeting with their counselor or a car owner bringing their automobile in for servicing.

The two systems are completely different implementations for two different companies and their rules are represented quite differently. However, several aspects of their architecture are quite similar In some ways these systems are similar to expert systems, although in each case the output is a structured report, rather than a diagnosis. Both systems represent knowledge (or business rules) as static declarative data that is traversed at runtime as users select from the available choices and enter data. However, an interesting characteristic of both of these systems is that there are multiple views (or "interpreters") over the underlying domain knowledge. The interpreters dynamically render the knowledge in various formats for different audiences, such as end users, third party information consumers, domain experts, and external business applications.

These systems can be re-configured at runtime by domain experts releasing new rules or revising rules, but the systems themselves do not generate new rules or modify their behavior dynamically based on their interactions with users. So neither of these systems is a self-adapting system. For one particular configuration of the knowledge base, the systems will behave deterministically, yielding identical results over time when two different users pursue identical paths.

2. Design Problems

In this paper, I will discuss a few design problems that one encounters while developing a business application that uses an independent representation of business rules. The primary advantage to such a system is that domain-specific rules may be maintained directly by domain experts. The problem is that rule development is still a type of software development, so standard development issues related to version control, debugging, and re-use still apply.

The material for this paper is drawn from several rule-based systems that have been developed by the author over the past 18 years. However, the primary focus is on current web-based architectures and the advantages of standard data formats, such as XML. Two very different systems have been built over the past two years, so I will propose some patterns for when one architecture might be more appropriate than another.

3. Editing Rules

3.1. Fat client vs. thin client

Many enterprises have been involved in attempts to "webify" business applications, so it is no surprise that tools for editing business rules are also faced with the choice of architectures for fat client vs. thin client. An installed application with local application logic is especially effective when:

(a) Response time must be faster than can be supported by a wide-area network.
(b) There is a very large rule base with strong inter-relationships requiring viewing, navigating, and modifying the entire graph of rules. For example, when editing the relationship between rules is just as important as editing an individual rule.
(c) A small number of domain experts are working on unrelated sets of rules.

A thin client implemented with a web interface can be useful where:

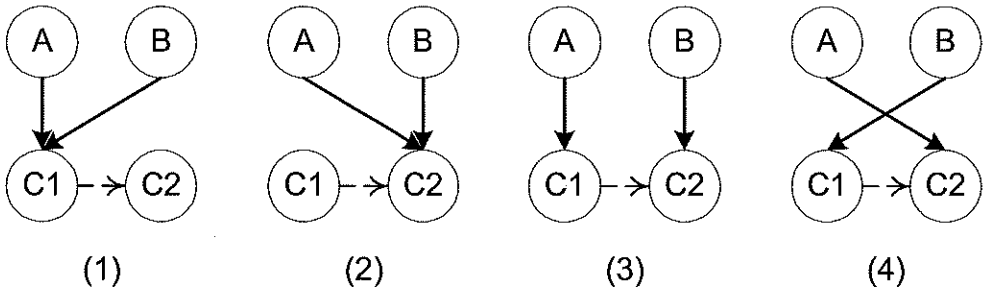
(a) Each rule is relatively independent of other rules.
(b) Editing a simple rule is more important and more common than editing the relationship between rules.
(c) Rules are relatively small in size, e.g. kilobytes, rather than megabytes.
(d) Domain experts are geographically distributed and there are large numbers of domain experts entering and editing inter-related rules, where the addition or change of one rule should be visible immediately to others.

3.2. Including rules by reference

Business rules can often be stated in a general form that is re-usable in several contexts. In order to re-use a rule, one might copy the rule into each context or one rule may simply refer to another rule. In general, inclusion by reference is desirable from the perspective of rule maintenance, because changes to one central definition will be reflected in each context in which the rule is used. However, inclusion by reference leads to its own set of problems for the developers of the rule editor, because the references must be maintained properly. For example, if rule A refers to rule B, can rule B be deleted? Also, the domain experts need to be able to view and trace the references. For example, before modifying rule B, an expert probably wants to know all of the rules that refer to B, so that a change to B will not cause unintended consequences. Therefore, the rule editor should provide some type of "where-used" query.

3.3. Versioning rules

When rules are re-used in multiple contexts, versioning becomes extremely important. Suppose we have two rules A and B that use sub-rule C and C is changed in some way from version 1 to version 2. There are four possible configurations that result:



Possible configurations after a rule is modified.

Configurations (1) and (2) are likely candidates to be managed automatically by the rule editor. Either the change is not propagated, as in (1), or else it is propagated automatically to all contexts, as in (2). If either (3) or (4) is a desirable state, then the domain expert probably needs to explicitly choose each environment to which the change is to be released.

For many systems, old versions of rules must remain in the system, in order to re-create previous lines of inference. Thus, even when a system is designed to update references automatically as in configuration (2) above, the original version of rule C may still be needed, in order to print old reports or re-create logic based on the original rule C.

3.4. Debugging rules

It is very useful to provide an interactive debugging environment for domain experts, so that they can see the effects of rule changes. In some development environments, such as implementing rules in Prolog, a full programming environment may be available to the domain expert. In other cases, it may be sufficient to provide a run-time simulator, so that the expert can observe the results of the rules when operating in an environment that is similar (or identical) to the end application. In this case, changes to the rules (or proposed changes to the rules) must be deployable and executable rapidly, so that the domain expert can quickly move from the rule editor to the execution environment and back again.

4. Storing Rules

4.1. Specifications of rules vs. implementation of rule behavior

Rules may be represented as specifications for behavior, rather than as implementations of behavior. This allows for different implementations that are appropriate for the particular

environment. The implementation may be functionally distinct, such as the difference between a rule editor operating on a rule and report generator using the contents of a rule. Or the implementations may be functionally equivalent, yet implemented with different programming languages for different environments.

This approach has been used for years with grammars, where a declarative representation of syntactic rules may be interpreted by any number of parsers. One parser might implement a top-down algorithm, while another might implement a bottom-up algorithm. However, both parsers can use the same statement of syntax.

4.2. XML as a common format for rule representations

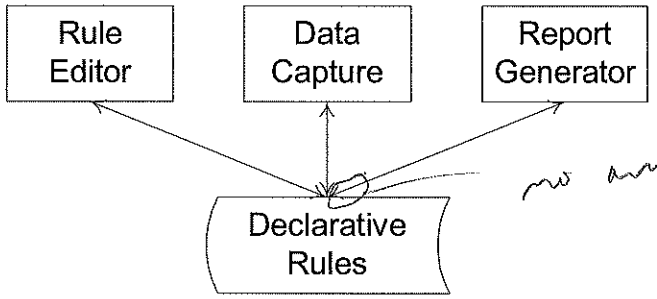
XML is becoming widely used as a common format for sharing data. Therefore, it makes sense to store domain-specific languages as declarative rules in XML. For performance reasons, it may be useful to compile the rules into objects or executable procedures. However, the XML is the base representation upon which other interpreters are based.

One of the problems with XML is that the overhead can be excessive for a very large number of rules. For example, I recently worked on a system where it was not uncommon to have a graph of 45,000 nodes where each node must contain the same set of tags, even though there might only be a small amount of unique domain information per node. In this case, the cost of using XML may become prohibitive. On the other hand, for rules that have less structure and more domain information per rule, the advantages of XML may exceed the cost of additional rule size, especially if there is a small number of rules.

5. Using Rules at Run-Time

5.1. Multiple interpreters

One of the advantages of storing rules as specifications (see Section 4.1) is that the same set of rules may be used by different interpreters. For example, one interpreter might use the rule to capture data from an end user, while another interpreter might generate a report from the instantiated rule that contains data. The following diagram shows three different applications operating from the same rule base:



Multiple interpreters operating on declarative rules.

5.2. Deploying rules

An important requirement for rule-based systems is to have a mechanism for deploying new or updated rules easily. Ideally, this becomes a procedure that can be managed by domain experts, who make the decision of when to release changes into the operational environment. As mentioned previously, versioning and configuration management may become a problem when there are many dependencies or if application requirements mandate the ability to replay a line of inference.

6. Conclusion

In this paper, I have discussed a number of design decisions for rule-based systems. The choice of one design over another will be determined by application conditions such as properties of the rules (size, dependencies, grouping), usage of the application (number of domain experts, location of experts, environment for end users), and requirements for using the rules (re-use in several applications, ability to replay lines of inference).

7. References

Berman, Gerald D., Richard N. Gray, David Liu, and James J. Tyhurst. 2001. "Structured radiology reporting: a 4 year case study of 160,000 reports." Presented at the Integrating the Healthcare Enterprise (IHE) Symposium of the Radiological Society of North America (RSNA) 2001 Annual Meeting, November 25 - 30, 2001.

Tyhurst, James J. 1986. "Applying linguistic knowledge to engineering notes." In S.C-Y. Lu and R. Komanduri (eds.), Knowledge-Based Expert Systems for Manufacturing (PED-Vol. 24). New York: The American Society of Mechanical Engineers. pp. 131-136.

Tyhurst, James J. 1986. "Natural language processing applied to engineering notes." Ultratech Artificial Intelligence Conference Proceedings (Vol. 1), Long Beach, California. pp. 2-199 to 2-211.

Tyhurst, James J., and Kerry L. Glover. 1988. "A menu-based interface for expert system rules." In Proceedings of the 2nd Annual Expert Systems Conference and Exposition (April 12-14, 1988). Detroit: Engineering Society of Detroit. pp. 203-210.

8. Contact Information

Author

Jim Tyhurst, Ph.D.
jim@tyhurst.com

Tyhurst Technology Group LLC
14335 S. Hawthorne Ct., Oregon City, OR 97045
503.632.7416
www.tyhurst.com

Tyhurst Technology Group provides consulting services to deliver effective business solutions based on the latest object-oriented architectures for enterprise applications. We have helped new or expanding software development teams to define development processes that meet their unique requirements, while developing software that meets the business goals of the client. This focus on business solutions ensures that the final deliverable of software will be of the highest value to the client organization.

Workshop Organizers

OOPSLA 2002 Workshop: Using Domain Specific Language to Drive Business Applications.
http://www.oopsla.org/ap/files/wor-26.html

Ali Arsanjani
arsanjan@us.ibm.com
IBM, E-business Application Development, Center of Competency.

Reza Razavi
razavi@acm.org
University of Paris, LIP6.