Combining different business rules technologies: A rationalization

Isabelle Rouvellou, Lou Degenaro, Hoi Chan
IBM T. J. Watson Research Center
{rouvellou, degenaro, chan}@us.ibm.com
Kevin Rasmus
Country Companies Insurance
Benjamin N. Grosof
MIT Sloan School of Management
Bgrosof@mit.edu
http://www.mit.edu/~bgrosof/
Dave Ehnebuske, Barbara McKee
IBM Austin

Introduction

This paper discusses the very early stages of a project being conducted at IBM research to explore how different rule technologies might be combined or enhanced to offer powerful business solutions.

People apply many meanings to the term "business rules." In its simplest form, a business rule is any statement that can be put in the form of "if then". Business rules can be implemented in a computer program by programming if-then statements in-line. This is very efficient performance-wise, but you have to know exactly what rules need to be place where. If the rules change at a later time, there can be high costs of maintenance. This approach also lends toward inconsistency because people create or change rules without a full understanding of all of the rules or how they might interact.

Over the past 30 years, many patterns and technologies have grown up around the development, management, and execution of business rules. Some of these include

of these categories (i.e., they include hybrid functionality).

Both technologies seem to have their own strength and we are faced with the following questions:

- ✍ Are there ways to combine each technology to make them more beneficial than they are being used alone?
- ✍ Are there enhancements that can be done to further leverage the rules technologies?

This paper will first provide some additional information on CommonRules and ABR. It will then explore these questions.

Common rules Overview

CommonRules has, as its foundation, a classic inference engine structure. Rules are supplied in the form of if-then. Data (i.e., facts) are provided to the system as input or through database/file interfaces. The inference engine determines the relationship between the supplied

dealing with the specific needs of individual partners. BRML is based on declarative logic programs, and in particular on an extended form of these called Courteous Logic Programs [5].

To further enable the dynamic addition of rules, CommonRules allows a flexible way of prioritized conflict handling for rules, based on the Courteous Logic Programs extension of logic programs. This enables rules to be specified and maintained in a more modular "common sense" style than usual. Additionally, for the sake of greater control and performance, CommonRules allows procedural attachments to business objects, i.e., to associate belief predicates with methods in general business objects in order to test conditions in the "if" parts of the rules or to perform actions after drawing conclusions.

ABR Overview

ABR (Accessible Business Rules) is a framework that enables enterprises to develop distributed business applications that systematically externalize the time- and situation-variable parts of their business logic as externally applied entities called "business rules".

The ABR approach starts with extending the object oriented analysis and design to include identifying the rules and the points of variability in a business process (i.e., points in a business process where the business logic is time- and situation-variable) [6]. Where benefits (reuse, reduced maintenance, consistency, etc.) can be achieved, the rule is externalized (separated to be managed separately and potentially more dynamically) and the point of variability is, during development, translated into a trigger point (i.e., a piece of code in an object method that interfaces with the ABR runtime to attach and execute business rules dynamically during application execution). A trigger point is the specific place from which externalized rules will be fired. A trigger point's primary job is to select which rules will be fired, based on some criteria called trigger point context. In all cases, the rules to be

logic is performed). The trigger point, rule selection, rule combination, data transformation, and administration capabilities taken together form a complete framework for externalizing and managing business rules [7] [8].

ABR has been developed as an object oriented (OO) framework and can easily be integrated with object oriented applications. Typical OO applications define different "object layers"-- each of which builds on what went before. The lowest level objects directly reflect the database data forming the relatively fined grained object model that experience shows is natural for fully capturing the required detail. As you move up the object model toward the user, the level of abstraction increases and business components are defined that are more easily understood by a business person; but their state data is only indirectly a reflection of data stored in the database. By attaching ABR rules at various layers of your application, you define them with various level of abstraction [8].

Some of the More Pertinent Differences in the Two Rule Approaches

As CommonRules and ABR are both hybrid approaches and have considerable flexibility, the following statements should be taken as generalizations rather than strictly true.

1) CommonRules expects to handle a rapidly changing rule base, including rules that are delivered from outside parties specific to a single transaction. ABR expects more "release oriented" rules where rules become effective or expire based on date and time.
2) CommonRules expects rules to be built on the fly and conflicts be dealt with by the rule system itself. ABR's administration system facilitates the identification of conflicts and expects most or all conflicts to be eliminated prior to release.
3) While both products support a "discovery" process for determining what rules are needed, ABR is more attuned to identifying rules during formal analysis and

require a few simple rules at any one point in the process.
7) The performance dynamics vary between the two products. A rule system must perform well to be found useful by most business people. ABR is more oriented towards static rules and as such sometimes has greater performance.
8) The manner of specifying a rule varies between the two approaches. As people vary in the way they express themselves, they also vary in the way they feel most comfortable in specifying rules.

When and How to Combine the Two Technologies

Next we identify some business situations or patterns where using a combination of the two approaches/ products appears to be valuable.

Innovative Options to Use the Two Approaches Together

1) At most points in a business process, only a few straightforward rules apply (ex. edits for the date field or the selection of which interest rate calculation applies). However, at a few very specific points, things can get very complex. The somewhat obvious response here is use ABR for the majority of the rules and use CommonRules when things get complex. Since ABR can wrap a CommonRules rule group as a single ABR Business Rule, it is easy for ABR to act as the trigger for CommonRules.
2) If different portions of a business process address different groups, each of these groups may be predisposed to a type of rule representation. For example, marketing may prefer CommonRules representation where accounting may prefer ABR .. Each product can be coupled to the portion of the process they best support.
3) Some portions of a business process may have well defined, tightly controlled, self-consistent rules that change at specific points in time. Other portions of the system may have highly dynamic rules where the encoding can then be exploited to get better performance when the prioritized rule set will not be changing often.

Use the Approaches/Products at Different Points in the Application Life Cycle

1) Some projects utilize a formal analysis and design approach. Other projects might utilize an iterative prototyping approach. Both rules approaches can be utilized in either of these project approaches. However, ABR expects a certain level of patterned design to be present. In the case of a project where you are making it up as you go, you may choose to start by capturing rules in CommonRules. At the point in the project where the architecture firms up, you may then wish to shift appropriate rules over to ABR.
2) If the rules for a project are not well understood, you may want to start with CommonRules to discover the interaction between rules. As rules interaction becomes clear, you may want to move rules to ABR to improve performance, reduce complexity, etc. The courteous compiler can help enable this move.
3) If an application starts of with ABR rules and over time the rate of rule change increases, or the complexity increases, or the need for dynamic behavior increases, you may want to shift rules to CommonRules.

Where Rule Systems Might be Enhanced

Data mining

1) Similar to the concept of case-based reasoning, you can analyze data to derive rules that can be implemented in either of these products. A promising area for this is personalization for web sites. You can analyze the navigational patterns of customers to derive rules that might better customize the experience of similar customer types.
2) Conversely, you can data-mine the rules themselves looking for patterns that aren't obvious from the rules themselves. This might lead to simplification of rules or improvements in performance.

This specific rules situation could be optimized and made particularly easy to use.
2) Businesses often have rule patterns that are specific to their company. The APIs for rule creation and administration can be externalized so a company can write front-ends specific to their rules patterns.

Authoring of Rules.

1) As stated earlier, people have preferences for how they state their rules. Furthermore, rules systems have differences in the level of ambiguity they will tolerate in rules. Many people who are comfortable in stating rules verbally become uncomfortable when forced to use specific rule syntax. A conversational natural-language system seems a possible way to facilitate development by business people. Here, the system interacts with the person to refine rules and arrive at a tolerated level of completeness and ambiguity.
2) Alternatively, UML (Unified Modeling Language) is commonly employed to model and specify a system. If currently availabl constructs or extensions of the UML language could be used for capturing rules, then a tool could extract rules from the model and implement them using a rules product. Many developers would find this convenient.

Conclusion

Rules products vary in their strengths and weaknesses. Combining rules approaches/products can leverage their strengths and minimize their weaknesses. We discussed several ways to synergistically combine two approaches/products with which we are familiar: CommonRules and Accessible Business Rules. Additionally, we believe there are still many opportunities to improve rule system functionality and enhance their business benefits. We hope these opportunities will be further explored, to the benefit of all.

References

Logic Programs in XML" (Nov. 3, 1999), in Proceedings of the 1st ACM Conference on Electronic Commerce (EC-99), edited by Michael P. Wellman, ACM Press.

[5] B. Grosof: "Prioritized Conflict Handling for Logic Programs", in the proceedings of the International Symposium on Logic Programming (ILPS-97), edited by Jan Maluszynski, MIT Press, Cambridge, MA, USA, pages 197-211.

[6] I. Rouvellou, L. Degenaro, D. Ehnebuske, B. McKee, K. Rasmus: "Extending Business Objects with Business Rules", in the proceedings of the 33rd International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 2000 Europe) http://www.research.ibm.com/AEM/abr_tools2000.html

[7] I. Rouvellou, L. Degenaro, D. Ehnebuske, B. McKee, K. Rasmus: "Externalizing Business Rules from Enterprise Applications: An Experience Report" in the Oopsla'99 Companion

[8] I. Rouvellou, I. Simmonds, D. Ehnebuske, B. McKee, K. Rasmus: "Business Objects and Business Rules" in "Business Object Design and Implementation: Oopsla '96, Oopsla '97, and Oopsla '98 Workshop Proceedings" Springer, ISBN: 1-85233-108-9.