Experience in Applying KRS to an Actual Business Problem

Rob C. Kremer
Computing Research
Shell Canada Ltd.
P.O.Box 100, Station M
Calgary, Alberta, Canada
shellcr@cpsc.ucalgary.ca

Abstract

This paper traces the application of a knowledge representation server (KRS) to an actual business problem in financial transaction verification and correction. The problem was first attacked using a traditional expert system shell, but the translation of the business rules turned out to be verbose and difficult to understand. The Z specification language was applied in order to try to gain a better understanding of the problem domain, and this showed the need for rules with exceptions. KRS has rules with exceptions, a powerful "programming with pictures" visual language, a link to hypertext facilities and other innovative tools that apply to the problem. KRS has been applied with some success as a prototypical system. It is hoped that the simplicity of a visual language such as KRS's may enable the domain experts to directly input the original business rules and to handle the ongoing updates.

1. The Problem

The company's financial transaction records are created at the various business sites, then are batched into the financial and accounting programs. However, some of the transactions are rejected because they contain inconsistent or erroneous data. The system provides an error code with the rejection. Somehow, these rejected transactions must be corrected and then resubmitted to the accounting programs.

Currently, the corrections are being done by a group of people who rely on a book of rules indexed on the error code accompanying the rejected transaction.

The task of correcting the transactions is difficult because it requires a large amount of knowledge about transactions; it's tedious; there may be more than one error in a transaction; and the rules are very difficult to read and interpret. It is often the case that a single transaction is cycled many times between the accounting system and the correcting group.

The rules are very long and unstructured, so it is very difficult for a non-expert to understand the rules. Once more, the rules are purely in English, so are open to misinterpretation. See figure 1 for an example rule.

1.1. What Do We Want?

It is hoped that an automated system, encapsulating the knowledge from the rule book, could be used as an expert assistant to the people responsible for transaction error correction. Ideally, rejected transactions could be feed directly to the new system, which could display transactions as an on-line form, with erroneous fields highlighted. Each highlighted field would be accompanied by suggested corrections. The user could either accept the suggestions or enter a new value, whereupon the entire transaction would be re-checked immediately (without cycling through the accounting system again). When the transaction is deemed to be correct, it would be returned to the accounting system. Simple, obvious errors could even be automatically corrected and returned to the accounting system without human intervention.

The rules of the business world change rapidly, so it is important to have a knowledge representation that is easy to understand and update. Since minor updates to the knowledge base will be done relatively frequently, we would like a system simple enough for the domain experts to update without extensive training. Given the complexity of the domain, this can probably only be done by hierarchically

structuring the knowledge base -- something most current expert systems shells don't do well. We draw an analogy here between hacking and unstructured programming.

INVALID FEDERAL TAX CODE - INVALID FTAX CODE

DESCRIPTION

A SALES INVOICE LINE WHICH REQUIRES A FEDERAL TAX INDICATOR SHOULD BE CODED WITH ONE OF THE INDICATOR VALUES SPECIFIED BELOW:

(A) FOR SALES OF AVIATION AND MOTOR GASOLINES (MAJOR PRODUCT CODES 101, 121, 211, 215, 221, 222, 224, 225, 228, 231, 235):

(1) IF THE PROVINCE TO IS '09' THE FEDERAL TAX CODE MUST BE 6.

(2) IF THE PROVINCE TO IS '02' AND THE TRADE CLASS IS '025' THE FEDERAL TAX CODE MUST BE 6.

(3) OTHERWISE THE FEDERAL TAX CODE CAN BE ANY OF 0, 2,4,6,7.

(B) FOR SALES OF TURBINE AND DIESEL FUELS (MAJOR PRODUCT CODES 141,142,322, 323, 325, 326, 327, 328, 329, 361)

(1) IF THE PROVINCE TO IS '09' THE FEDERAL TAX CODE MUST BE 7.

(2) IF THE PROVINCE TO IS '02' AND THE TRADE CLASS IS '025' THE FEDERAL TAX CODE MUST BE 7.

(3) THE FEDERAL TAX CODE CAN BE 8 IF THE DOCUMENT DATE IS PRIOR TO SEPTEMBER 3.1985.

(4) IF THE PRODUCT CODE MAJOR IS 361, THE TAX CODE MUST BE 2 OR 7.

(5) OTHERWISE THE FEDERAL TAX CODE CAN BE ANY OF 0, 2, OR 7.

(C) FOR SALES OF HEATING OILS (MAJOR PROJECT CODES 301, 302, 304, 305, 306,321, 324, 362) THE FEDERAL TAX CODE MUST BE 7 OR 8. EXCEPTIONS:

(1) PRODUCT CODE 362300 SHOULD BE TAX CODE 2 OR 7.

(E) FOR DISPLAY ADVERTISING (PRODUCT CODES 426-996, 426-997 AND 426-998) THE FEDERAL TAX CODE MUST BE 7.

(F) FOR AIRPORT FEES (MAJOR PRODUCT CODE 030), THE FEDERAL TAX CODE CAN BE 0, 2, 4, 6, OR 7. THE FEDERAL TAX CODE CAN BE 8 IF THE DOCUMENT DATE IF PRIOR TO SEPTEMBER 3, 1985.

(G) ON NON-EXPORT CARDLOCK SALES, (TRANSACTION 5670, OR 5600 WITH TRANSPORTATION CODE '44') FOR PRODUCTS 4XX, 5XX, AND 9XX, THE FEDERAL TAX CODE MUST BE '0'.

(H) FOR ALL OTHER SALES THE FEDERAL TAX CODE CAN BE 0,7 OR 8.

Figure 1.: A business rule.

2. History

We first tried to encode the informal rules in a commercial expert system shell. While this could be done, it turned out that a single informal rule expanded to an incredibly large number of shell rules. Furthermore, the shell rules where very difficult to understand and to map back to the original informal rules. Both due to predicted size of the knowledge base, and the predicted difficulty of maintenance, this avenue was rejected.

The next approach was to use the Z specification language to try to get a better understanding of the rules and get an idea of what kind of tool could implement a solution. Z turned out to be extremely useful to experiment with different methods of implementation, but it could not, of course, give us an implementation. After many paper-and-pencil experiments, it became clear that the rules could, for the most part, be rewritten as a hierarchy of rules with exceptions. Thus we can say things like "if A then X unless B then Y", which means "if A is true then do X, but if B is also true then we should do Y and not

X". Although this has a straightforward interpretation in simple rules[1], it is extremely hard for a user to understand (and maintain) the rules when the exception hierarchy becomes large[2].

3. KRS as the Prototype Platform

Now that we were aware of what we needed, we happened upon KRS [Gaines, 1991d], a knowledge representation server which not only contained rules with exceptions but several other features that applied to the problem. KRS contains a powerful visual language that we believe would be useful for the domain experts to input the rules directly themselves with minimum training. It also deals with sets and "negative sets" nicely, which is useful in the domain. Furthermore KRS can be easily linked to a hypercard stack where we can easily implement an on-line form interface.

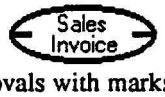
3.1. Visual Language

The visual language allows the user to create graphics to represent the knowledge domain. The graphs are amazingly easy to understand (with a little training), and should significantly ease the maintenance problem over textual representations. The visual language is translated to KSSn, where it can be used by the KRS server. There is also a visual language that can translate from KSSn to the visual language. It is important to note that the visual language is not stored as KSSn and then reconstructed, but is actually stored such that the image is reconstructed exactly as the user had created it. It is critical to let the user arrange and group the symbols in a way that make sense to the user, not the machine. Simple and powerful graphical tools are provided for moving, grouping, cutting-and-pasting, etc.

3.1.1. Graphical constructs

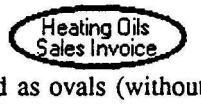
The graph is composed of nodes with connecting arcs. There are six different nodes: primitive concepts, concepts, individuals, rules, roles, and constraints. Every node is and icon labeled with its name. There are two different arcs: directed arcs[3] and undirected arcs. Most of the constructs will be briefly explained below. For a more detailed description see [Gaines, 1991a]

Primitive Concepts



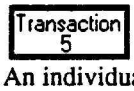
Primitive concepts are represented as a ovals with marks on either side. A primitive concept is a schema or class definition.

Concepts



Non-primitive concepts are represented as ovals (without marks). A non-primitive concept is fully defined by its relationship to other concepts, a non-primitive is *recognized* when other entities comply with its definition.

Individuals



Individuals are represented as rectangles. An individual is an object with identity.

Rules

---

[1] "if A&~B then X, if A&B then Y"

[2] When there are multiple exceptions to A or when B, in turn, has exceptions.

[3] The directed arc is highly overloaded.

Rules are represented as rectangles with rounded corners. The input concept to the rule is the antecedent, and the output concept is the conclusion of the rule. In other words, if the input concept holds, then the output concept is asserted to hold. Multiple input arrows are considered to be disjunctive.

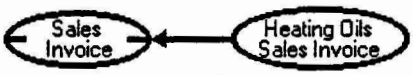
Roles

Roles are represented as the label text with no surround at all. Roles are used, in this case, to represent *attributes*, or *fields* of the transaction records.

Constraints

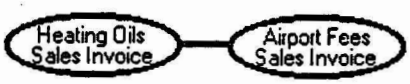
Constraints are represented as rectangles with rounded corners and marks at either side. Constraints are usually used with roles to type the role or for concept recognition.

Subsumption



A directed arc between two concepts indicates that the concept at the tail is subsumed by the concept at the head. This is the isa relationship, or can be thought of as inheritance.

Disjointness

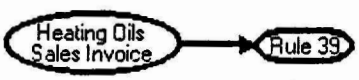


An undirected arc between two concepts indicates that nothing can be asserted as being a member of both concepts.

Instantiation

A directed arc from an individual to a concept asserts that the individual exhibits the properties of the concept. In other words, the individual is an object of the schema.

Antecedent



A directed arc from a concept to rule indicates the concept is an antecedent to the rule.

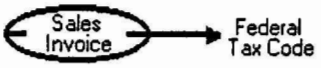
Conclusion

A directed line from a rule to a constrained role indicates that the role is constrained if the rule is fired.

Exceptions

A directed arc between two rules indicates that the tail rule is an exception to the head rule.

Attributes



A directed arc from a primitive concept to a role indicates the role is an attribute of the concept (schema).

Types

When a role is attached to a primitive concept, a directed arc from the role to a constraint constrains the value that can fill that role, and so defines (or partially defines) type information about the role. In this case the role *federal tax code* is a label (string) with a cardinality of one.

Recognition

When a role is attached to a primitive concept, a directed arc from the role to a constraint is used for concept recognition. In this case, an *invoice* is recognized to be a *heating oils sales invoice* iff the *product code major* is 301, 302, 304, 305, 306, 321 or 324.

Multiple occurrences of the same node (same node type and label) in a graph are deemed to be the same entity, so the user may define a concept in one place and use it in another without problem. This enables grouping in complex constructs without an inordinate amount of arcs clogging the visual effect.

3.1.2. Comments

The user may place comments anywhere on the graphic surface just by creating a node (usually a role, since there is no surround) with no arcs connected to it. This is often very convenient for later maintenance when the rules have to be changed due changes in the business world.

3.1.3. Visual representation of a business rule

The visual language translation of the business rule given in figure 1 is shown in figure 2. This rule is easier to understand than the informal rule, and is not ambiguous. Another advantage is that the ambiguities of the informal rule have been removed. For example, when the *major produce code* is 141 and the *province* is '09' the informal rule can be interpreted as either allowing or disallowing *federal tax code* of 8 (see clauses B1 and B3); whereas the visual representation makes it clear that a federal tax code of 8 is not allowed (see lower left concepts *turbine & diesel fuels 09* and *new turbine & diesel fuels*).

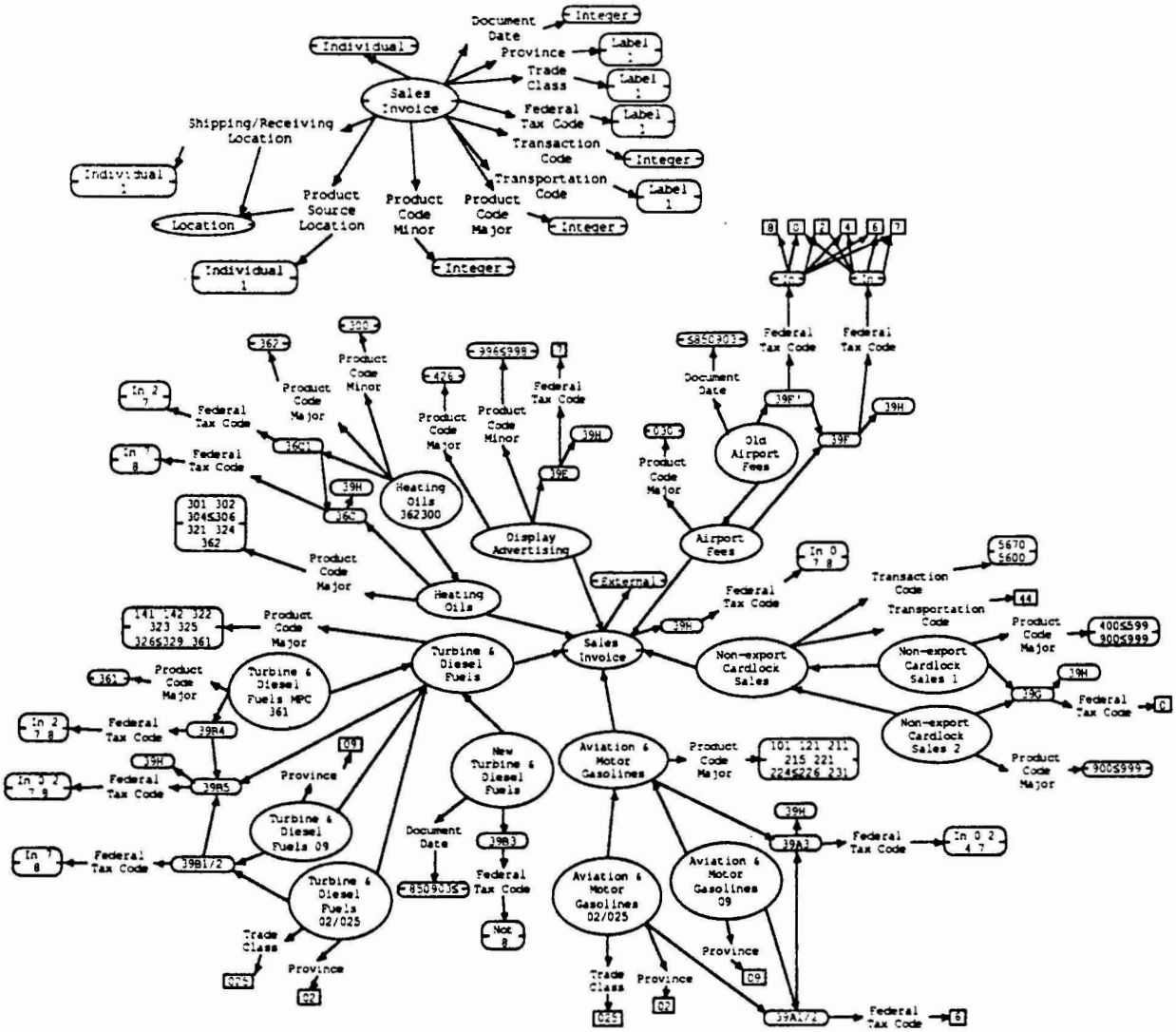


figure 2: Visual language representation of the rule in figure 1

3.2. Hypercard Link

A hypercard stack can communicate with a KRS process by opening a IPC channel to it and passing messages through the channel. Hypercard can send any message that is legal KSSn syntax. Likewise, return messages are the text expected from normal KSSn text interface. The hypercard stack must parse the return messages appropriately, but this is not difficult since the KSSn output has been designed to be easy to parse.

Hypercard is used for an on-line forms interface (see figure 3). In the case of the prototype, the user can fill in the form[4] and then click on the Check Case button to have the KRS run on the record. Any erroneous fields will be highlighted and a suggestions field opened to the right of the field. The user is free to make corrections and recheck the record.

[4] In the actual system, the form would be pre-filled from the accounting system.

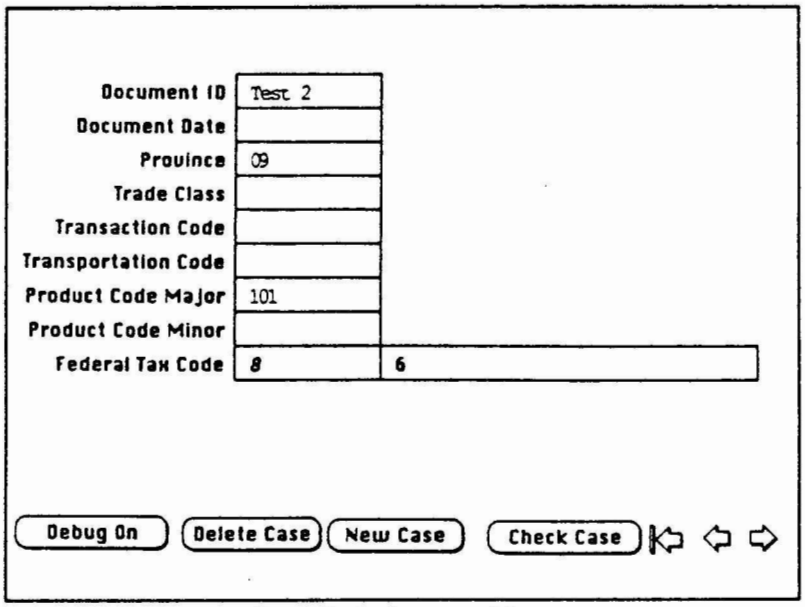


figure 3: the hypercard form

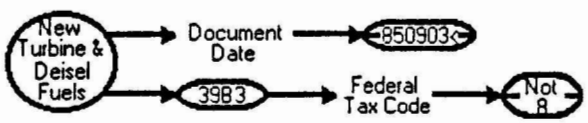
3.3. Sets and Ranges

The example informal rule points out the need for handling of sets with a set notation that can specify both discrete values and ranges of values. For example, almost every major (lettered) clause in the rule refers to the *major product code* field, and a list of ten or so discrete values that it may contain for the clause to apply. Clause G refers to *major product codes* in the 400's 500's and 900's -- awkward to list as discrete values. The former clause (for clause A) can be represented with a constraint with the label "In 101, 121, 211, ... 235", while the latter (clause G) can be represented with a constraint with the label "In 400≤599, 900≤999". The discrete notation and range notation may be mixed freely in a single constraint.

3.4. Negative Sets and Ranges

Another useful feature of KRS is its handling of negative sets. Clause B3 of the example rule is an example use. Clause B3 says that any transactions with *document dates* prior to a certain date are allowed to have *federal tax codes* of 8. Without negative sets, this clause would not have its own identity, but would result in two versions of the other clauses, one each for before and after the date.[5] This lack of direct correspondence makes for both an encoding problem for the novice user and a maintenance problem later on.

With negative sets, we can encode clause B3 by recognizing newer-than-the-date transactions, and adding a rule that newer transactions cannot have a federal tax code with value 8:



None of the other clauses are affected, and even if we add new clauses the date rule still applies to them.

3.5. Constraints

[5] For trivial example, see the visual translation of clause F (airport fees) in the upper right corner of figure 3. For a more telling example, try to encode clause B (turtine and diesel fuels) without using the **not** operator.

Constraints are used both with concepts for recognition and in conclusions of rules to constrain the values the roles may take on. The difference here is not superficial. Concepts constraints can be seen as *passive* constraints where we assume the data (value in the role) is correct and we are merely recognizing the concept. Rule constraints are *active* constraints where we make no assumptions about the correctness of the data but will enforce it. In the former case the data is driving the processing of the knowledge base; in the latter case the knowledge base is processing the data.

4. The Future: An Implementation Platform

KRS is currently implemented on the Mac, and the company demands that PCs, VMS Vaxes, and IBM mainframes be used exclusively. So KRS, in its present form, cannot be used as an implementation platform. If further work on the project proves successful, some appropriate subset of KRS could be translated to one of the other hardware platforms. Since KRS is implemented in C++, this should not be too onerous a task.

5. Conclusion

More work needs to be done to prove that KRS can represent all facets of this problem (the rule book looks like a telephone book), but I am extremely optimistic that it can be done. The final implementation will require a few add-ins like links to remote databases.

KRS has shown itself to be a very useful tool for developing knowledge bases for domains that are based on reasoning with exceptions. KRS's visual language has shown itself to be surprisingly useful for knowledge base development and maintenance. The visual language's most valuable attribute is enabling the user to give hierarchical structure and informal grouping (a problem with a lot of current expert system shells) to the knowledge base by allowing knowledge bases to be spread over multiple files and user placement of icons.

References

[Gaines, 1991?a] Gaines, B.R. "An Interactive Visual Language for Term Subsumption languages." ???

[Gaines, 1991b] Gaines, B.R. "Empirical Investigation of Knowledge Representation Servers: Design Issues and Application Experience with KRS." ???

[Gaines, 1991?c] Gaines, B.R. "Integrating Rules in Term Subsumption Knowledge Representation Servers." ???

[Gaines, 1991?d] Gaines, B.R. "Organizational Modeling and Problem Solving using an Object-Oriented Knowledge Representation Server and Visual Language." ???