White Paper:

The Representation and Execution of Business Operation Rules

Table of Contents

Abstract

The automation of business operation or process rules is one of the few ways for business to respond quickly to changing environments, and to enforce management decisions uniformly across organizations. This paper outlines the problems that businesses face when attempting to automate these rules and describe how business can use InfoSapient to successfully and rapidly deploy and maintain business operation rules.

Keywords: business rules, fuzzy logic, rules server, databases, application logic, business process, e-Commerce.

The Problem

Business can draw pictures of the way information flows through its organizations, illustrate for management the sequence of actions the organization will follow under given conditions, and graph its management structure.

But, in a very real sense, these business process illustrations and management hierarchy diagrams do not cover a very important aspect of the organization: the set of rules that determine how a business operates, e.g. rules that prevent, cause, or suggest things to happen[1]. These business operation rules are often 'soft' rules, e.g. rules that are almost never written down, but are commonly 'rules of thumb' (also known as 'experience') that reflect the current thinking of human administration to manage and maintain the organization.

Responding to Changing Business Conditions

To promptly react to changing conditions, business must be able to settle on what action to take to implement those decision(s) as rapidly as possible. However, it is well understood that making decisions within an organization is usually much simpler than implementing them. The reasons for this are varied, but many times are due to one or more of the following factors:

1. The difficulty of explaining the decisions, communicating the impact and meaning of those decisions and insuring that the decisions are implemented in a way that reflects the original management thinking *across the entire organization,* especially in diverse and remote locations.
2. The speed at which the decisions become implemented becomes problematic as the size of organization grows. Because of the speed of changing environments and the time lag to implement the reaction, many times the operation rules are implemented only in time to become at best irrelevant, at worst, possibly harmful to the operations of the organization. Alternatively, they may simply never be implemented at all.
3. Changing business operations to meet new challenges means that business must show workflows and business processes in terms of their relationships to soft rules. Simultaneously, it must express the process of acquiring, maintaining, and enforcing the appropriate 'soft' rules *and how they affect*

those workflows and processes. Table 1 illustrates the problems and challenges of business complexity today.

Business Characteristic	Emphasis
Functional Complexity	Complex Business Rules; Many Interfaces, Business Objects, Work Flow
Operational Complexity	Distributed Locations, N-Tier Applications, Heterogeneous Platforms
Breadth of Initiative	Enterprise-Wide, Internal and External Customers, International Scope

Table 1: Existing Large Enterprise, Future Medium and Large Enterprise Requirements[2]

Current Situation

Typically, if business has automated business operation or process rules, they exist in one or more of the following locations: either within business applications (computer code) or databases (implemented as triggers). Even worse, many of a corporation's business rules exist today in legacy programs, and may be duplicated or overridden within departmental programs written in languages such as Visual Basic.[4]

Because of these applications being unknown to audit teams, or offsite in remote locations, or simply lack of time, departmental applications/rules may not be inspected by corporate data process auditors for corporate-wide compliance.

Problems with Current Implementations

There are major difficulties with either or both of these approaches, namely:

1. What happens to your current rule base if you change development platforms, deployment languages, or databases?
2. Where does the implementation of the rule go within diverse applications? How do you avoid 'repeating the exercise' for enterprise wide applications?
3. If the rule is implemented, what effects will that have on existing processes? How can you test this change to not only make sure that the rule does what you want in terms of process management, but does not introduce unintended side effects? Due diligence requires complete application regression testing when changes as the above are implemented. Regression testing is expensive and time consuming. How does our organization cut this time down?
4. Where do you find this rule in case it needs to be changed? How do you give this responsibility to someone else with a minimum of training?
5. How do you render operational semantics, such as 'near profitability', 'very costly', into application code?

The Solution

This paper suggests that many of the above problems in the automation of business rules can be mitigated or eliminated through the melding of several unrelated but useful technologies.
These technologies are:

- Executing business rules through the use of a business rules server, and
- Expressing the semantics of business rules using fuzzy logic.

The Business Rule Server

The concept of a business rule server offers several advantages over conventional architecture. Instead of implementing business rules embedded within application logic and/or database triggers, it provides a means to provide both business logic functionality, and yet separate it from application code. This permits easily viewable, easily changeable rules with a minimum of impact on existing application logic or databases.

Precedents

As a concept, dividing application services has many precedents. First, the entire foundation of object-oriented analysis and design dictates that the benefits of object collaboration are much greater than designing an application from a monolithic perspective. The simpler an object is in terms of functionality, the greater likelihood for that object to have reuse in future applications ~ conversely, the more complex the functionality of an object or program, the likelihood of reuse is greatly decreased. It has been demonstrated a myriad of times that is much easier to understand collections of small, even tiny, collaborating 'programs' working together than it is to understand the workings of a program consisting of several thousand lines of C, COBOL, FORTRAN, or Assembly.

Nowhere has this thinking been more articulated than in the separation of business logic from the user interface, and the separation of business logic from its persistence mechanism.

The Model View Controller paradigm (created in the early 1980's) separates business logic from the user interface through the mediation of a 'controller'. The use of this was justified on the basis that if business logic changed, developers should not have to change the user interface and vice versa ~ if the user interface changed one should not have to change the underlying business logic. This architectural concept or pattern has been one of the most successful ideas within modern IT development and has seen implementations within 3270/ASCII character based environments to modern thin-client web-based applications.

The next major separation of functionality came with separation of business logic and its persistence. (Saving the state of the application to a database.) Indeed, this has probably been even more profitable for major IT companies as they have all developed persistence frameworks and mechanisms. IBM and its competitors have

all developed extensive persistence products for custom application development and off the shelf products.

Fuzzy Logic

Unlike classical logic, which requires a deep understanding of a system, exact equations, and precise numeric values, Fuzzy logic incorporates an alternative way of thinking, which allows modeling complex systems using a higher level of abstraction originating from our knowledge and experience. Fuzzy Logic allows expressing this knowledge with subjective concepts such as 'very hot,' 'bright red,' and a 'long time' that are mapped into exact numeric ranges.

Fuzzy Logic has been gaining increasing acceptance during the past few years. There are over two thousand commercially available products using Fuzzy Logic, ranging from washing machines to high-speed trains. Nearly every business application can potentially realize some of the benefits of Fuzzy Logic, such as performance, simplicity, lower cost, and productivity.

Fuzzy Sets

The fundamental unit of computation within fuzzy logic is the fuzzy set. The following illustration represents the fuzzy set "Tall People" as a function of Height:

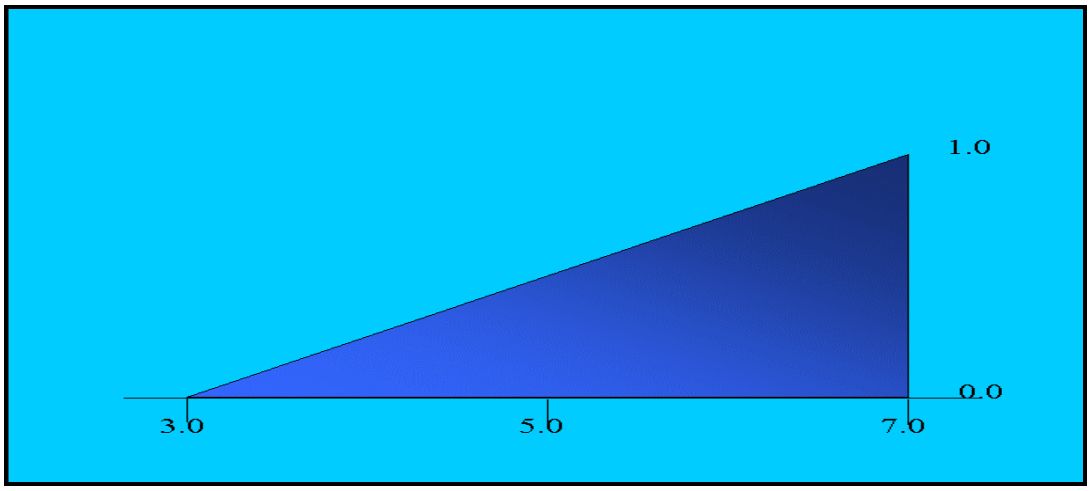


Figure 1:'Tall People' As a Function of Height

Note that the X-axis represents the domain in feet, and the Y-axis represents the degree of membership represented from 0% to 100% within the set "Tall People".

The fuzzy set Tall People has three properties: the low domain, the 100% membership point, and the high domain. The low domain is the start of the set on the X-axis, the 100% membership point is the point on the X-axis that corresponds to 100% membership within the set, and the high domain is the end-point on the X-axis. In the example set Tall People, the 100% membership point is also the high domain.

Typically, the 100% Membership point is located halfway between the low domain and the high domain, but may be located anywhere within the set. Obviously:

Low Domain ≤ 100%Membership ≤ High Domain

Person	Height	Degree of Membership within 'Tall People'
Billy	3' 2"	0.041667
Yoke	5' 5"	0.53667
Drew	5' 9"	0.6875
Erik	6' 1"	0.770833
Kareem	7' 2"	1.0

Table 2: Membership in the Fuzzy Set Tall People as a function of Height

In our example, a person that is 3 feet or less, would have 0% membership within Tall People. Anyone 7 feet or greater (Kareem), would have 100% membership within Tall People.

Finally, 5 feet would have 50% membership within Tall People. Constructing a relationship, this way also means that we can immediately classify the degree of membership of an element by taking one measurement, and reading off the degree of membership from the hypotenuse.

Fuzzy Surface Modifiers

One of the most powerful features of fuzzy logic is the ability to express everyday concepts such as 'very', 'slightly', 'extremely' into clearly defined and quantifiable terms. These terms, known as "Hedges", are used to modify the surface shape of the fuzzy set, and hence change the membership for all the elements within the set. In each of these cases, the term represents some sort of mathematical operation applied to the hypotenuse of the set.

Hedge Name	Operation Applied
Very	$f(y) = y^2$
Extremely	$f(y) = y^3$
Slightly	$f(y) = \sqrt{y}$
NOT	$f(y) = 1 - y$
(There are many others)	

Table 3: Hedge with Corresponding Operation

Using the example "Tall People" from above, the hedge 'VERY' is applied to it:

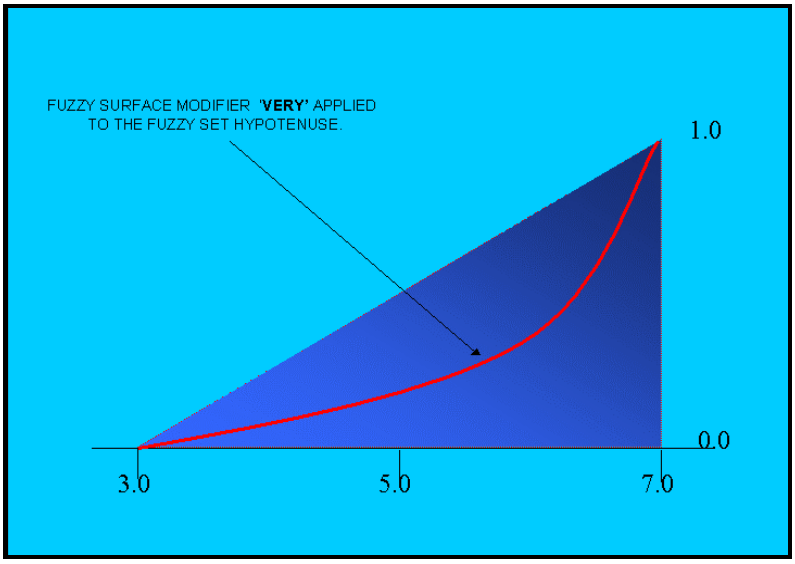


Figure 2: Application of the Hedge 'VERY' to the Fuzzy Set 'Tall People'

Now a person having a height of 5.0 no longer has a membership of 0.5 within the set of tall people. He or she now has an approximate membership of 0.25 in the set of VERY tall people. Before, a person that was 6' 1" had a membership of 0.77 in the set of tall people. That person now has a membership of 0.594 (just a little bit over 50% membership).

The next example demonstrates what happens when the slightly hedge is applied:

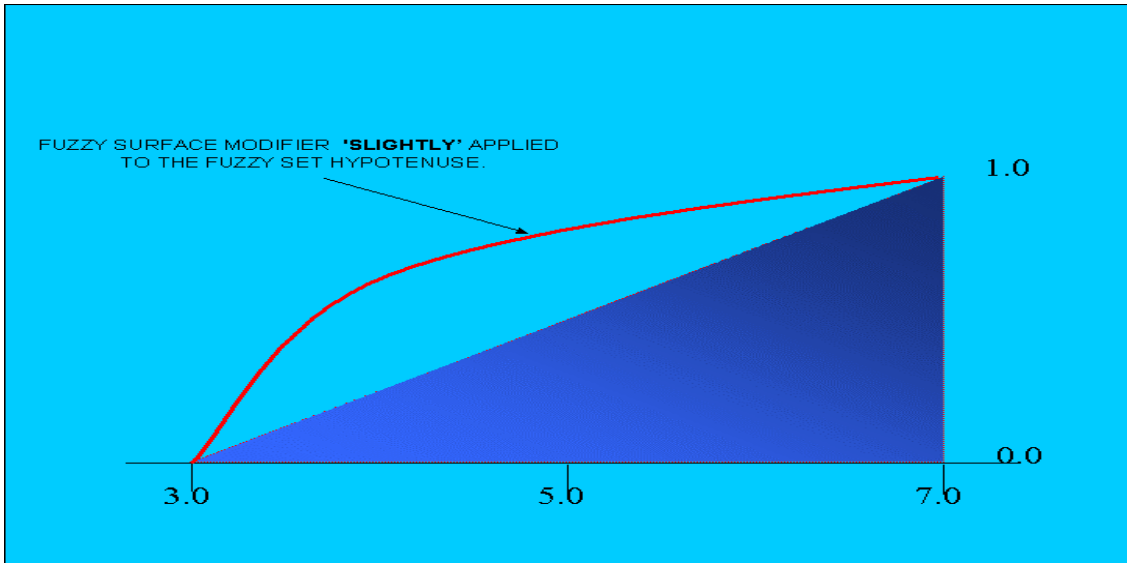


Figure 3: The Hedge 'Slightly' Applied to the set 'Tall People'

In this case, a person having a height of 5.0" and a membership of 0.25 in the set of very tall people would now have a membership of 0.75 in the set **'slightly'** tall people.

Finally, we have the example of applying the two hedges **'not'** and **'very'** to the set of tall people:

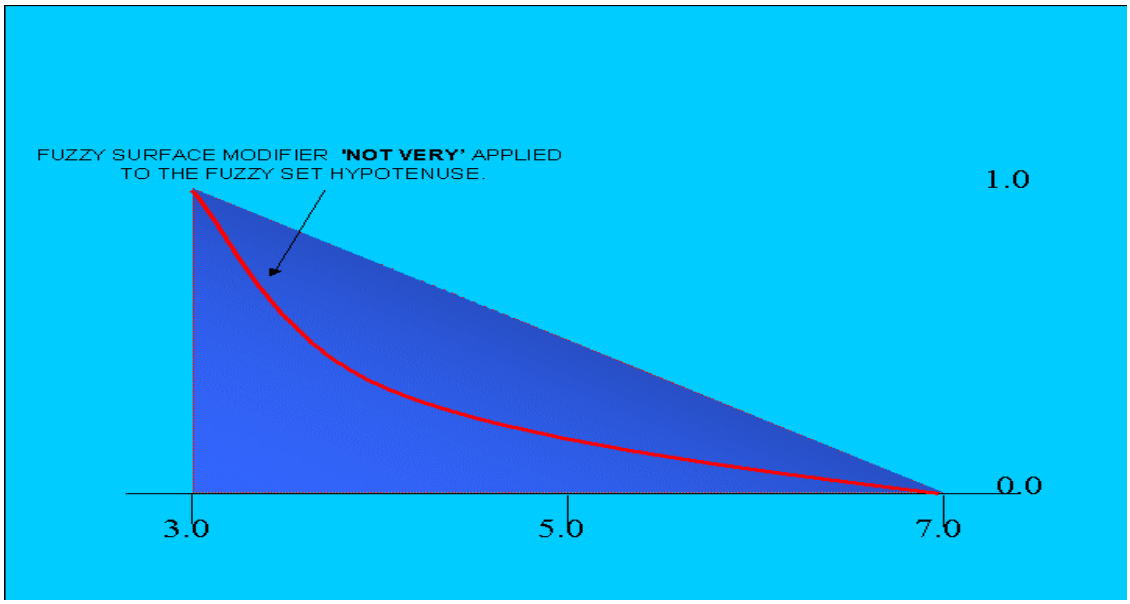


Figure 4: The hedges 'not', 'very' applied to the set 'Tall People'.

Once again, the power of linguistic expressiveness is shown by fuzzy logic. With the use of hedges and fuzzy sets, we can now describe precisely the concepts of *very profitable, slight loss, etc.* We have always had these concepts available to us in human communication, and now we have them available for computational use as well.

Fuzzy Attributes

The next example uses fuzzy sets to describe a continuum over a domain.

A CFO wishes to describe corporate profitability. She may use the terms "negativeProfitability," "zeroProfitability," and "highProfitability" to describe varying states within the general notion of profitability. Certainly, she knows what are meant by those terms, but she may encounter some difficulty communicating them to the rest of the organization, and probably a great deal of difficulty trying to implement those terms within application code.

However, fuzzy logic provides a very intuitive mechanism for doing just that: simply construct multiples of sets, each overlapping the next, so that the entire domain is covered. In this way, attributes or properties are described using fuzzy set notation. (For the mathematically inclined, this is a *power set.*)

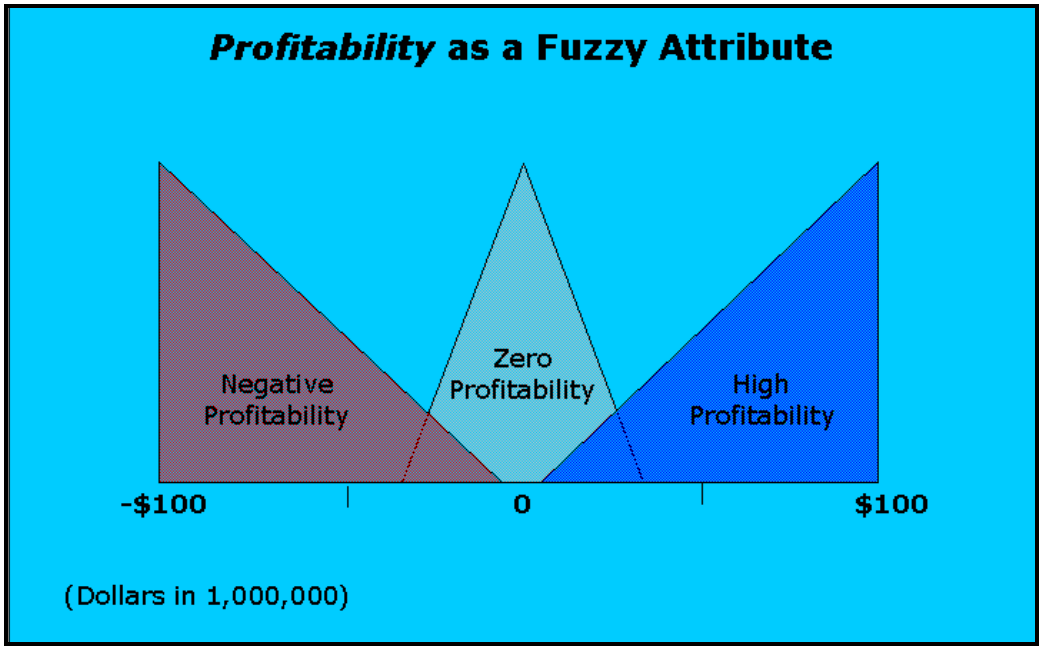


Figure 3: Attribute 'Corporate Profitability' as a Fuzzy Power Set

Set Name	Low Domain	100% Membership	High Domain
negativeProfitability	-100,000,000	-100,000,000	-30,000,000
zeroProfitability	-60,000,000	0	60,000,000
highProfitability	30,000,000	100,000,000	100,000,000

Table 4: Sets and Low Domain, 100% Membership, and High Domain Values

Logic Operations

The standard logic operator definitions in fuzzy logic are:

Logic Operator	Operation
μ (not a)	1.0 – μ (a)
μ (a and b)	minimum (μ (a), μ (b))
μ (a or b)	maximum (μ (a), μ (b))

Table : The NOT, AND, and OR Operators

(μ: Degree of Membership. Degree of Membership can also be thought of as the degree of *truth* or *falseness*. Some researchers in fuzzy logic have explored many other uses or interpretations of the AND, OR, and NOT operations, but for the purposes of this paper, these are sufficient.[3])

Assume for a moment that the degree of membership for a = one, and b = zero. This means that the μ (a *and* b) = **zero**, and that μ (a *or* b) is **one**. *These values are the same as they are for Boolean logic.*

The Extension Principle

If you plug just the values zero and one into these definitions (just as we did above), you get the same truth tables as you would expect from conventional Boolean logic. This is known as the *extension principle*, which states that the classical results of Boolean logic are recovered from fuzzy logic operations when all fuzzy membership grades are restricted to the traditional set {0, 1}. This effectively establishes fuzzy sets and logic as a true generalization of classical set theory and logic. In fact, by this reasoning, all crisp (traditional) sets *are* fuzzy subsets of this very special type; and there is no conflict between fuzzy and crisp methods.

Logic Operation Examples

However, let us assume that μ(a) = 0.34 and that μ(b) = 0.78. In fuzzy logic now, the ***and*** operation: μ(a *and* b) = 0.34. The ***or*** operation is: μ(a *or* b) -- now 0.78.

Now that you know what a statement like "profitability is low" means in fuzzy logic, how do you interpret an everyday statement like:

> **If** (Credit is badCredit or Income is lowIncome) **then** Application is denied?

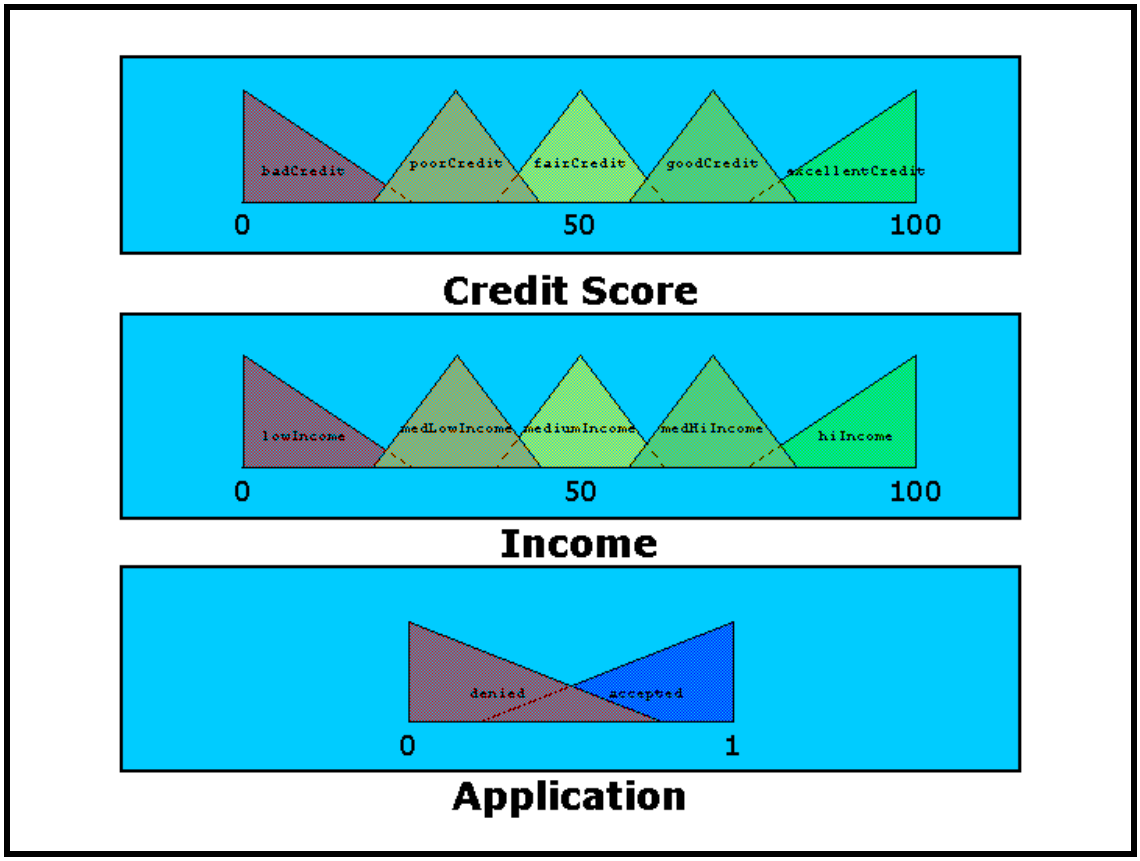


Figure 4: The Fuzzy Attributes 'CreditScore', 'Income', and 'Application'

This rule has two parts: the premise (also called the *antecedent*) and the consequent. The premise consists of the clauses "Credit is bad" and "Income is low". The consequent (or what is concluded if the premise is true) is: "Application is denied."

In Boolean logic if the premise is found to be true, then the consequent or conclusion must be true. However, in fuzzy logic, the consequent is true *only to the degree that the premise is true.*

Assume in our example that the domain CreditScore is 23 and Income is 87. Looking at the attribute CreditScore, we find that the score 23 lie within both of the sets 'badCredit' and 'poorCredit', but that has a slightly greater membership within badCredit (≅0.23) than poorCredit (≅0.10).

Our applicant has an income of $87,000, which falls outside of the domain of the set 'lowIncome', so the membership of this clause is zero.

Using the fuzzy logic operators above, we take the maximum of the clauses:

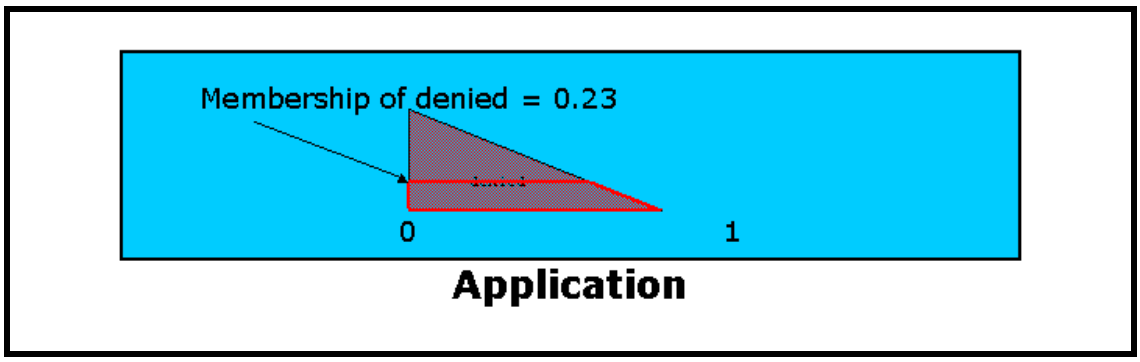
$$\mu(\text{rulePremise}) = \text{Maximum } ((\mu(\text{badCredit})=0.23), (\mu(\text{lowIncome})=0.0))$$

μ(rulePremise) = 0.23

Since the consequent's membership is now set to the membership of the premise, the remaining item is to decide as to where on the Application domain does the answer lie. In this particular case, it is not so important that we obtain a single value on the domain of Application, simply that we determine whether the application is approved.

Therefore, we take the first point on the domain within the set 'denied', which has a membership of 0.23. In this case, that point is 0.

In summary, we have now been able to translate concepts such *as CreditHistory is badCredi*t into quantifiable and computable terms.



How Fuzzy Logic Performs Conflict Resolution

Many times in life, experts disagree on what a solution to a problem is. (Imagine!) Fuzzy logic performs extremely well in these cases, in that it can take apparently mutually exclusive decisions, and reconcile them.

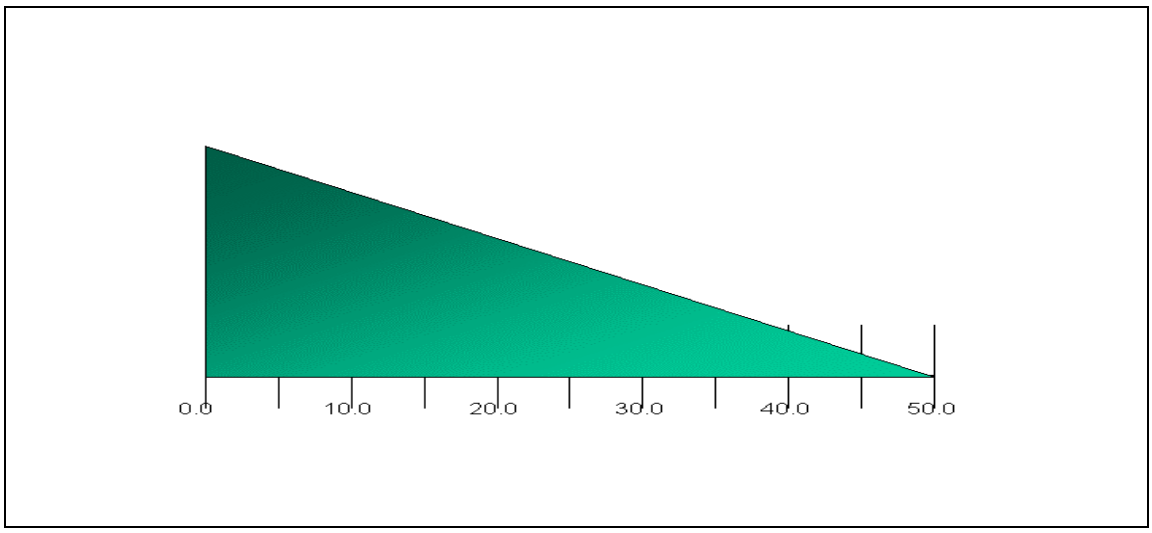
Consider the (admittedly oversimplified and contrived) case of a hypothetical company, the management of which is currently considering the pricing of a new product. Each of the VP's have a different idea on what the price should be, based on their particular perspective and expertise.

VP of Sales	"Our price should be low!"
VP of Finance	"Our price should be high!"
VP of Manufacturing	"Our price should be greater than our manufacturing costs."
VP of Marketing	"Our price should be around our competitions price."

In each of these cases, an agenda is advanced that may conflict with some or all of the agendas advanced by the other VP's.

Assumptions
Let us assume that the product price must be between $0 and $50.00 (US), that the competition's price for a similar product is approximately $23.00, and that our manufacturing department has found that we can produce our product for $19.00.

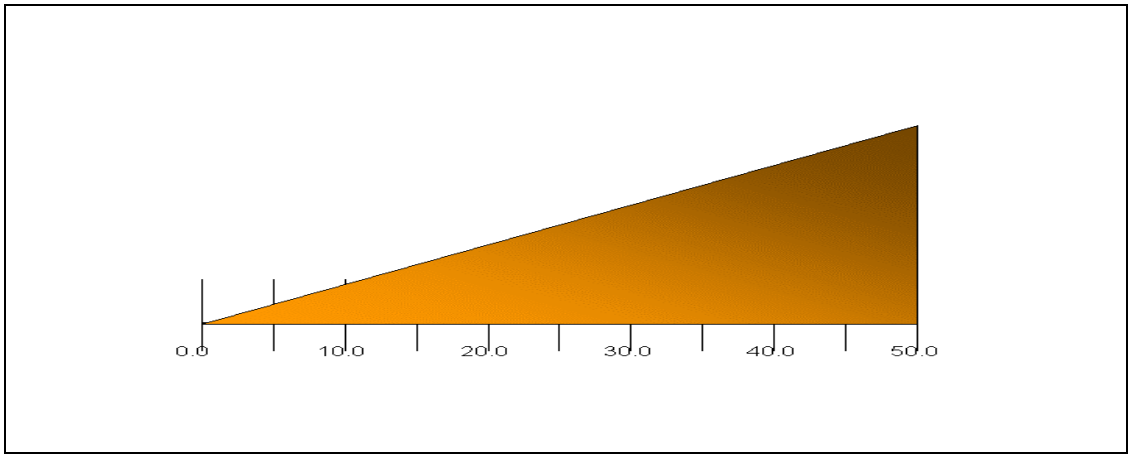
We can represent the VP of Sales pricing desire with the following:



VP Sales: 'Our price must be low.'

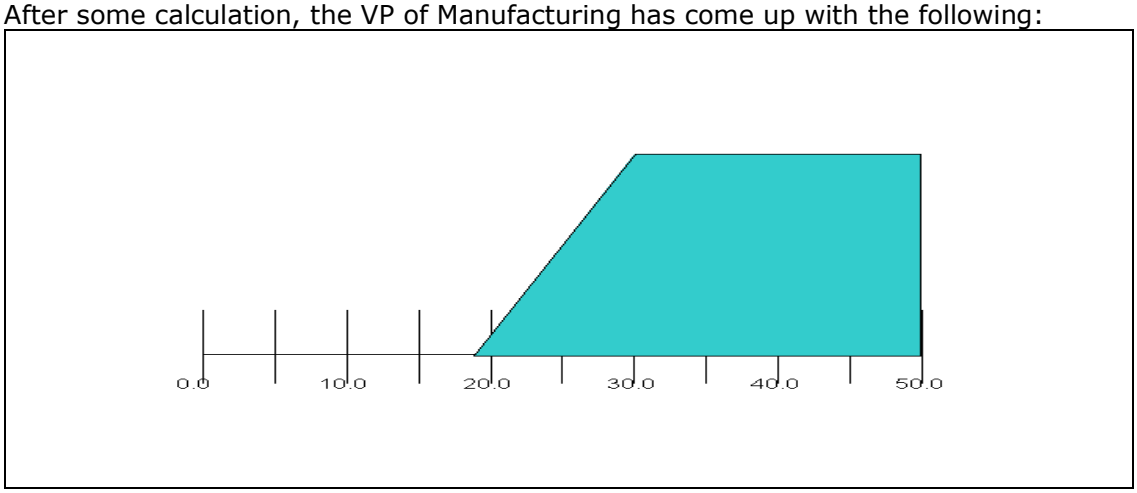
Note that $0.0 dollars represents the (very unreasonable) 100 % membership for "low price".

The set of the VP of Finance looks the same but reversed, and has the equally unreasonable 100% membership of $50.00.



VP of Finance: 'Our price must be high'

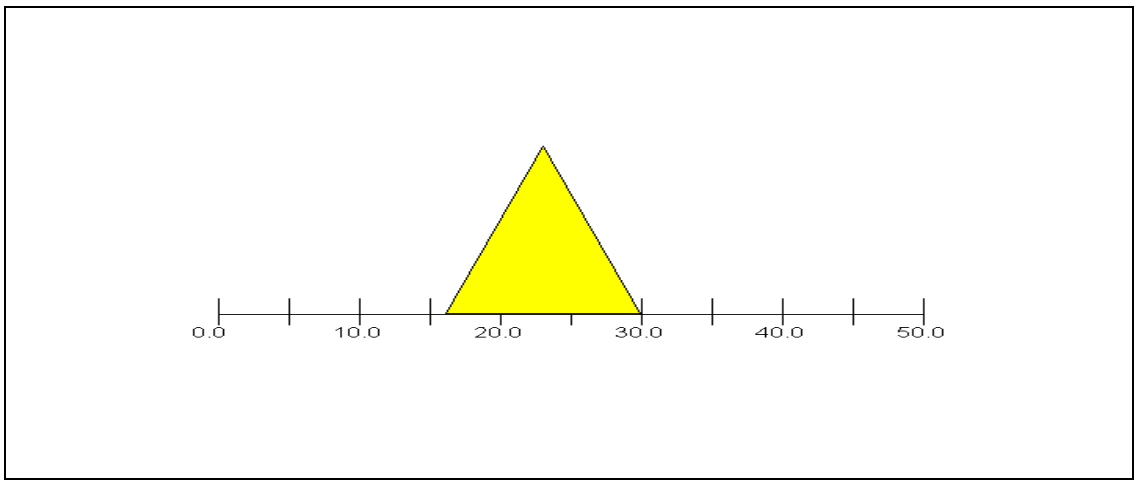
After some calculation, the VP of Manufacturing has come up with the following:



VP Manufacturing: 'Our price must be above our costs'

Actually, the VP of Manufacturing will be satisfied with anything above cost recovery, which we have identified as $19.00/unit. Note that the low domain of 'above costs' is at 19.00,where it has a 0% membership, and grows to 100% membership around $30.00/unit.

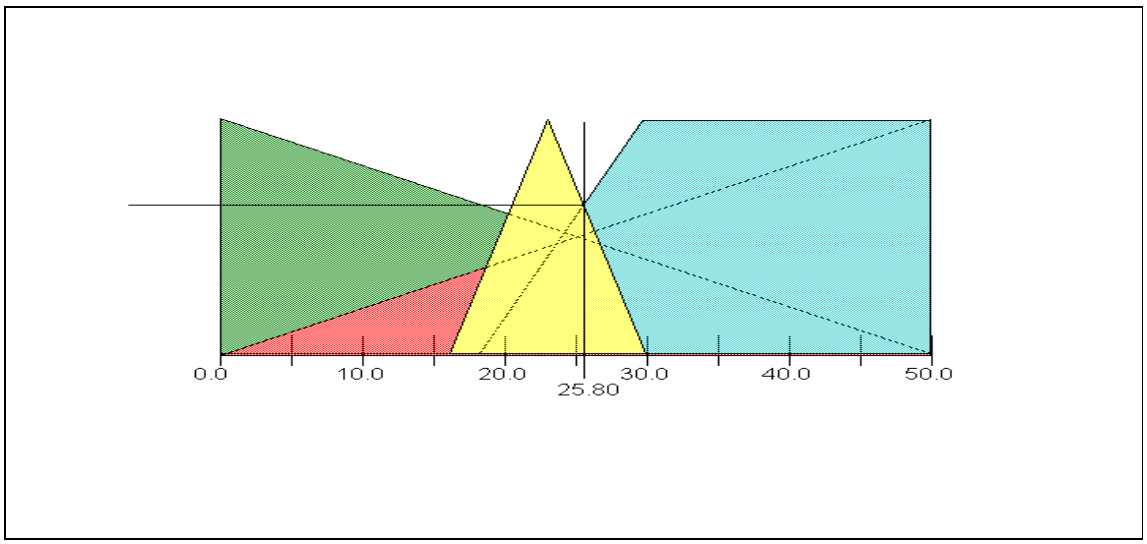
Finally, the VP of Marketing weighs in:



VP of Marketing: 'Our Price Should Be Around Our Competition's'

We can represent the number '23' (our competition's unit price) as a fuzzy set, with 23 representing the 100% membership point, '16' representing the low domain of the set, and '30' representing the high domain. (The low and high domains were chosen arbitrarily.)

The way that all of these agendas are reconciled is by superimposing them together, and picking the intersection that is as high as possible for each set, and still includes each set. This results in a "defuzzified" product price of $25.80, and a degree of membership that is approximately 75%. (In this case, the degree of membership represents the 'goodness of fit' of the solution.)



Product Price

The primary point of this example is that the solutions from each proposal contributed equally to the final decision, *even though two of the suggested prices were mutually exclusive.*

Specifications

The proposed solution integrates the concepts of a business rule execution engine or server, and the use of fuzzy logic to describe natural language terms into computable notions.

Standards

InfoSapient is 100% Java™ based, and has both servlet-JSP and stand-alone services. It has not officially been certified as 100% Java™ Compliant, albeit it is written as such. The knowledgebase uses XML for knowledge and rules representation. (A *knowledgebase* is a collection of knowledge expressed using some formal knowledge representation language.)

Components

The components of InfoSapient comprise a knowledgebase creation tool, and the actual rules engine or server. In this case, a knowledge representation language similar to colloquial English has been created and implemented within InfoSapient.

Rules Engine

The rules engine uses 'Backward Chaining" to determine which rules apply to the current hypothesis to test. Backward chaining is defined as: "An algorithm for proving a goal by recursively breaking it down into sub-goals and trying to prove these until facts are reached." (Facts are goals with no sub-goals that are therefore always true.) Please see Appendix A for an example of how backward chaining works.

InfoSapient uses widely understood compiler technology (Symbol Tables) to build consequence->premise references and maintain these references in memory.

After determining which rules apply, fuzzy logic is used to actually decide *how* rules are executed and employed when conflict resolution is required.

Loosely Coupled Connectivity

When deployed, the InfoSapient rules engine is used 'loosely coupled' to other applications. This means that the only interface or communication mechanism between the client application and the rules engine is a pipe or socket. The InfoSapient application runs on a server, using a dedicated port (typically #7010, but configurable). The client application opens a connection to the rules server, and writes the following information: the goal and other required information. The rule server will then withdraw a thread from its thread pool and begin running the knowledge session. The result including the domain value and the degree of membership is written back to the client application, and the thread is placed back into the thread pool.

InfoSapient can be deployed as a tightly coupled application extension as well, since it is Java based and is extensible.

Discussion

How well does InfoSapient answer some of the problem with implementing business rules listed earlier? What are the benefits and drawbacks to using InfoSapient?

Benefits

1. Provides automatic rule conflict resolution.

2. Provides much finer grain of control over when a decision is made, and under what circumstances that decision is made.

3. Rules are directly expressed in English language-like constructs, and are human readable.

4. Rules are completely separated from the database and application logic. When a rule is changed, the application and database logic does not need to be changed/tested.

5. Rules are stored using XML syntax – permits other applications to examine and use them if necessary.

6. Since rules are externalized from the application and database, they are easy to change, permitting company operations to proceed quickly when economic or other conditions change.

7. User interface permits testing the rules under varying conditions before actually deploying the rules to production.

8. Rule execution is very fast, compared with other algorithms.

What InfoSapient Currently Cannot Do

*(*Planned for implementation in future releases.)*

1. InfoSapient does not have 'forward chaining' as a solver implemented. This means that it cannot search its current fact base for any rule that meets that fact, then execute the rule. (This type of solver is very useful in product configuration and process control.)*

2. Currently, each knowledge session is stateless, e.g. it knows nothing about what previous sessions determined, and cannot save its state as input to other consultation sessions that could use the information.*

3. InfoSapient has no mechanism of spanning knowledgebases, e.g. using the output of one knowledge domain as input to another.*

4. By current design, the rule base currently cannot access external data. This means during the consultation session, the 'client' must supply the goal to be solved, and all supporting information as well.*

5. Currently, it is unable to represent and execute unconditional rules, e.g. rules of the form "Our price must be high". (Unconditional rules represent bounds on the domain of the solution.)

6. The rule syntax does not permit calculations, i.e. if ((a + b) is greater than m) then x; or executing external programs, scripts, or methods on external objects.* In other words, the client must execute any other external object, or script, based on the results from the consultation session.

7. InfoSapient does not handle reasoning about multiple instances of the same type within the knowledgebase, for example, it cannot handle rules of the form **ifAny:**, **ifAll:,** or **ifNoneOf:*** (These types of rules are very powerful, and plans are to implement these types in future releases.)

   > **ifAny** (car has a completed workup) and (car has no outstanding damage) and (car is not due for a overhaul) then car is eligible for rental.

8. The InfoSapient knowledgebase does not currently use a type hierarchy for knowledge representation.* (For all practical purposes, you might think of a SP knowledgebase as a Singleton, and its attributes and rules as the Singleton's instance variables and methods. Using a hierarchy of types for knowledge representation is very powerful, and plans are to implement this in future releases.)

When Should InfoSapient Be Used?

InfoSapient is intended to be as generic as possible. It is not intended to be any one industry specific solution, albeit there may be better industry fits than others. Please see "Industry Applications" for a full listing of possible industry applications.

1. InfoSapient should be used anywhere there is a requirement that the decision making process consider 'shades of gray' – finer grains of control *as well as yes or no*. *Please note – INFOSAPIENT is eminently capable of making Boolean decisions as well.* (See page 10 and 11 for a full discussion.)

2. Whenever there is a need to change rules rapidly, and not impact existing applications. E.g. pricing decisions, web site personalization, work flow decisions, etc.

3. Implementing business rules within a new application suite.

When Should InfoSapient Not Be Used?

InfoSapient should not be used where required rules specifications fall under any of "What InfoSapient Currently Cannot Do".

In addition, existing legacy applications are frequently not a good candidate for SP ~ even though SP uses a simple socket connection, for connection to in-place applications. This is because extensive rework is typically required to remove current business rule logic, and replace it with calls to the SP rule engine.

Industry Applications

The following are a few potential industry applications of the InfoSapient product.[4]

Health Care

- The health care industry can use business rules to create an enterprise view of clients.
- Care providers can gain access to a complete view of all interactions and treatments, no matter where the patient enters their network.
- Greater client satisfaction results by not having to repeat information at different locations.
- Healthcare staff can concentrate on providing the highest quality care.
- Companies can offer value-added services, such as customized electronic bulletins about health news and new services relevant to specific client concerns, and targeted discounts on over-the-counter pharmacy items.

Securities

- Applications based on business rules allow brokerage houses and financial services to make business more profitably using automated platforms to handle basic customer inquires.
- Customers can get unique and tailored phone menus based on their demographics and transaction history, so the company can do targeted selling and boost customer satisfaction and retention.
- Using rules, companies can now route callers to the right representative or agent, ensuring that experienced staff always handle high-valued, experienced investors.
- The company could use rules to route trades automatically to post-trade systems and to validate securities transactions against SEC regulations. This reduces fines and improves customer satisfaction.

Manufacturing

On web sites:

- Business rules enable applications to personalize web pages and tailor menus to present each customer with a unique view of the company and its product and services.
- Intelligent agents assist customers in selecting the right products.
- Rules cut costs and increase delivery speeds by automatically routing transaction data from frontline web applications to back office systems.

On sales support applications:

- Business rules enable companies to dynamically create quotes and proposals based on current market segments and customer history and total value to the enterprise.
- Allow finely targeted cross selling and up selling based on dynamic market segmentation and customer value assessments.

Streamlining build to order:

- Rule-driven applications enable manufacturers to deliver highly customized products faster and more cost-efficiently.
- Rules allow sales support systems to instantly view product availability and production capacity so sales representatives can negotiate prices, discounts, and schedules.

Retail

Rule-driven applications allow retailers to selling 1-to-1 in real-time:

- Rules help retailers do personalized selling in stores and online.
- They also allow web sites, store kiosks, and sales support applications to dynamically generate customized product and service information and offers based on customer profiles, purchasing history and total value to the retailer.
- Rules permit up selling and cross selling based on this information.

Rules turn knowledge about customers into sales:

- Allow retailers to use what they learn about their customers through point-of-sales data capture and data mining on the sales floor.
- Turn knowledge into new pricing, promotions, policies, and other business policies into fast, simultaneous deployment across all selling systems.

Insurance

- Business rules enable insurance companies to speed up underwriting and claims approval processes through automated systems.
- Enable software applications to exceed the best estimating capabilities of today's quotation systems.
- Use applications to underwrite and rate applications with the same expertise as senior underwriters.
- Confidently sell binding insurance policies over the net, with accuracy and consistency.
- Customer satisfaction increases with shrinking cycle times.
- Profitability increases with decreasing labor costs.

Potential Other Applications

- Matching PO's to Invoices
- Pricing decisions
- Out of Trend Analysis – Fraud Detection
- Risk analysis
- Workflow decisions
- 'Fuzzy' SQL
- Product Configuration

Bibliography

1. *GUIDE Business Rule Project, Final Report*; Nov. 6, 1995.
2. *SIMethod Training*; Internal IBM Presentation; 1999.
3. *Fuzzy Logic -- Frequently Asked Questions, hosted by Carnegie Mellon University;* 1993
4. *"Ruling a Self Service World"*; www.blazesoft.com; 2000.
5. *"Play by the Rules", Byte Magazine; 1997.*
6. *"Workflow-based applications";* IBM Systems Journal, 1997.
7. Earl Cox; *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems, Second Edition;* 1998; AP Professional

Appendix A – An Example of Backwards Chaining

An example of backwards chaining in order to reason about whether a customer can rent a car:

```
If (CustomerCredit is bad )then Rental is declined;
If (CustomerCredit is good) then Rental is approved;

  // Better'n cash
if (CustomerPaymentMethod is AMEX) then CustomerCredit is good;

  // Drug Runner?
if (CustomerPaymentMethod is cash) then CustomerCredit is neutral;

  // WE DO NOT TAKE CHECKS....
if (CustomerPaymentMethod is check) then CustomerCredit is bad;
```

In order to determine whether a rental is approved, the rules engine is asked what is the value of CustomerCredit. It first looks to see if CustomerCredit has been previously computed. If so, it directly determines Rental from this information.

If CustomerCredit has not been previously computed, it determines that there are three rules that determine CustomerCredit from their consequent. The rules engine then looks to directly to compute CustomerCredit from those rules. If it cannot, it looks to see if there are any rules that determine the *premise* (what is the value of CustomerPaymentMethod?) of those three rules. If there are, it looks further, if not, it asks the user 'What is the value of CustomerPaymentMethod'?

Since it knows that it can directly determine the value of Rental from CustomerCredit, after knowing what the value of CustomerPaymentMethod, it computes the answer and sends it back to the user.

Note that if this were the extent of our rule base, and the value of 'cash' is given, the rules engine cannot determine from *these rules alone,* whether or not to approve the rental. More rules would need to be incorporated to help determine the value of CustomerCredit.

Appendix B – InfoSapient Rule Syntax in BNF Form

Production Rule		Expansion
conditionalRule	::=	<IDENTIFIER> premise consequent ";"
premise	::=	<IF> clause
consequent	::=	<THEN> attributeClause (<ELSE> attributeClause)?
clause	::=	"(" clause ")"
	|	((expr | attributeClause) operator (expr | attributeClause)
		(operator (expr | attributeClause))*)
expr	::=	"(" expr ")"
	|	attribClause
attribClause	::=	id ("is" | "are") (hedgeCollection)? (nLiteral | id | restrictionHedge)
hedgeCollection	::=	(hedge)+
restrictionHedge	::=	("increased" | "decreased")
hedge	::=	("about" | "above" | "after" | "around" | "before" | "below" | "closeTo" | "definitely" | "extremely" | "generally" | "mostly" | "must" | "near" | ("negative" | "negatively") | "not" | ("positive" | "positively") | "roughly" | "should" | "slightly" | "somewhat" | "very" | "inVicinityOf")
operator	::=	("and" | "boundedAnd" | "cosineNot" | "meanAnd" | "meanOr" | "or" | "productAnd" | "productOr" | "sugenoNot" | "thresholdNot" | "yagerAnd" | "yagerNot" | "yagerOr")
nLiteral	::=	<FP_LITERAL> (nLiteral is a "numeric literal", e.g. it permits rules of the form: "if foo is extremely 5.0;" where "foo" represents a previously defined attribute.)

id	::=	<IDENITFIER> (rule name, attribute name, or set name)

Appendix C – InfoSapient DTD

```
<!ENTITY % DDbE_Out.en SYSTEM "DDbE_Out.en">
%DDbE_Out.en;
<!ELEMENT alt-conseq-clause (component,(is|are), component)>
<!ELEMENT are EMPTY>
<!ELEMENT attrib-clause
((component,(is),(hdg)*,component)|(component,(are),(hdg)*,component))>
<!ELEMENT attribute (set)*>
<!ATTLIST attribute
 type          CDATA   #IMPLIED
 prompt        CDATA   #IMPLIED
 description   CDATA   #IMPLIED
 initial-value CDATA   #IMPLIED
 id            CDATA   #REQUIRED
>
<!ELEMENT attributes (attribute)+>
<!ELEMENT author EMPTY>
<!ATTLIST author
 first-name CDATA #IMPLIED
 date-written CDATA #IMPLIED
 last-name CDATA #IMPLIED
>
<!ELEMENT premise (clause | expr)>

<!ELEMENT clause  ((expr, opr, expr)
              |(expr)
              )>

<!ELEMENT expr    ((attrib-clause)

              |(expr,( opr, attrib-clause )*)
              )>



<!ELEMENT component EMPTY>
<!ATTLIST component setname CDATA #IMPLIED
                attrname CDATA #IMPLIED
                literal  CDATA #IMPLIED
>
<!ELEMENT consequent (pri-conseq-clause,alt-conseq-clause?)>

<!ELEMENT hdg EMPTY>
<!ATTLIST hdg
 id CDATA #REQUIRED
>
<!ELEMENT is EMPTY>
<!ELEMENT kb (author,attributes,rules)>
<!ATTLIST kb
```

```
 correlation-method (product | minimum ) "product"
 resolution-method  (average-maximum | maximum | centroid) "centroid"
 implication-method (addAggregation | min-max ) "min-max"
 description CDATA #IMPLIED
 id CDATA #REQUIRED
>

<!ELEMENT opr EMPTY>
<!ATTLIST opr
 id CDATA #REQUIRED
>

<!ELEMENT pri-conseq-clause (component,(is|are), component)>

<!ELEMENT rule (premise,consequent)>
<!ATTLIST rule
 id CDATA #REQUIRED
>
<!ELEMENT rules (rule)+>
<!ELEMENT set EMPTY>
<!ATTLIST set
 cmp-membership CDATA #IMPLIED
 type CDATA #IMPLIED
 id CDATA #REQUIRED
 hi-domain CDATA #IMPLIED
 low-domain CDATA #IMPLIED
>
```