DeTerminator

Report on a
Decision Support System and Tool-Kit
using the ProFuSE method

by Ronald C.F. Antony, Axel G. Merk

Submitted in partial fulfillment of the requirements for the degree of
Master of Science
in the
Department of Computer Science at Brown University

May 1992

Leslie P. Kaelbling (Advisor): Leslie P Kaelbling 14 May 1992

0 Abstract

In the following report, we describe a decision support system and tool-kit that combines advantages of Bayesian Inference, rule-based systems, object-oriented decomposition and adds the capability of dealing with multivalent data. The purpose of this is to obtain a system capable of dealing with three notions that are important for making decisions under uncertainty. We call these the dimensions of property, probability and risk. Furthermore, we describe the algorithms and their properties that are fundamental to this decision support system. In a last step, we empirically test our model tool on the domain concerned with trading in financial markets and comment on some of the considerations that are essential in the creation of our test model.

BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-92-M15

''DeTerminator''

by

Ronald C.F. Antony and Axel G. Merk

Table of Content

0 Abstract 2
1 Definition of Objectives 5
2 Introduction 6
3 Design of DeTerminator 7
3.1 Overview of Decision Support System 7
3.2 Choice of Expert System 7
3.2.1 Paradigms in review 8
3.2.2 ProFuSE 10
3.2.2.1 Mathematical Interpretation of the ProFuSE Method 10
3.2.2.2 Approximation of Standard Paradigms 10
3.2.2.3 Custom Functions 13
3.2.2.4 Symbolic versus Numerical Evaluation 13
3.2.2.5 Notes on Computational Complexity 13
3.2.2.6 Isolation of Error Sources 15
4 Implementation of DeTerminator 16
4.1 Implementation Language 16
4.2 Notes on Implementation Structure 16
4.2.1 Program Structure 16
4.2.2 Data Structures 19
4.2.3 Algorithms 19
5 Empirical Test of Modelling Tool 21
5.1 Application Domain 21
5.1.1 Constraints due to Information Available 21
5.1.2 Constraints due to Know How 21
5.1.3 Application 22
5.2 Modelling the Knowledge Base 22
5.2.1 Requirements 23
5.2.2 Notes on the Knowledge Base 23
5.2.2.1 Indicators 23
5.2.2.2 Rule Functions 29
5.2.2.3 Joint Probability Density Functions 29
5.3 Preprocessing of Indicators 29
5.3.1 Implementation of Indicators and Musical Scores 29
5.3.2 Caching of Values 30
5.3.3 Date and Time 31
5.4 DeTerminator on the Test Stand 31
5.4.1 The Crash 32
5.4.2 Overall Performance 32
5.4.3 Dealing with Ambivalent Information, a Special Case 33
6 Conclusion 35
7 Outlook 36
8 Acknowledgments 37
9 Appendices 38
9.1 Bibliography 38

Table of Figures

Figure 1: ProFuSEe Rule page 9
Figure 2: derivation of joint probability density function page 11
Figure 3: xor, and, or as ProFuSE rules page 12
Figure 4: linear dependencies as ProFuse rules page 12
Figure 5: computational complexity, typical case page 14
Figure 6: computational complexity, worst case page 14
Figure 7: program structure page 16
Figure 8: sample input with inspector page 17
Figure 9: sample rule page 18
Figure 10: sample network page 18
Figure 11: data structure for input and rule page 19
Figure 12: application structure page 20
Figure 13: macd page 25
Figure 14: macd decision tree page 26
Figure 15: NYSE excerpt page 27
Figure 16: on balance volume decision tree page 28
Figure 17: on balance volume page 29
Figure 18: Scores page 30
Figure 19: Score page 30
Figure 20: rule structure page 31
Figure 21: expert opinion, crash 1987 page 32
Figure 22: a sample strategy page 33
Figure 23: hussein rule page 33
Figure 24: expert opinion, hussein 1991 page 34

1 Definition of Objectives

Our objective is to design a modelling tool that is capable of evolving over time into a potent decision support system. It is a major concern of ours to provide means such that knowledge modelled in our system retains clearly defined semantics. This means that we want the system to have more than some fuzzy notion of likelihood or a vague description of characteristics. Instead, we aim at separating the dimensions of property, probability and risk.

Furthermore, we want a consistent means of representing knowledge that is powerful enough to express arbitrary rules rather than just a few special cases.

Last, but not least, we want a system that scales well and is capable of performing in an on-line, near real-time environment.

On a subordinate level, we want to explore applications of time-oriented data structures and exploit the advantages provided by leveraging on object-oriented technologies.

We will empirically test our system on an example taken from trading in financial markets. It is based on information that is readily available to us and does not take advantage of all information that might be needed to achieve its task best.

ALPHONSE KARR 1808–1890
Plus ça change, plus c'est la même chose.
The more things change, the more they are the same.
Les Guêpes, Jan. 1849. vi

2 Introduction

After having studied a wide range of topics related to economics - and in particular finance - as well as computer science - with an emphasis on artificial intelligence -, it seemed to be a logical choice to combine these areas in a Master's project. Also, when we visited a conference on *Artificial Intelligence Applications on Wall Street* [AI/WS 1991], it seemed to us that many of the approaches presented were rather arbitrary and incohesive.

It appears to us that making decisions under uncertainty requires that the decision-maker has knowledge not only about what properties are valid at a given moment or are expected to be valid at some point in the future, but rather has an idea on how likely every possible value of a property may be at some point in the future. The reasons for this are intrinsic in the definition of mean values and expected values that are traditionally used to act under uncertainty. As one of our economics instructors, Professor Jerome Stein, phrases it, a clock that stands still on six o'clock is right on average - although it is almost always wrong.

To get a clear picture of a situation, a probability density function over the range of possible values helps a lot in assessing different possible actions. With a probability density function, we capture two out of three important dimensions: property and probability because for each possible value, the property can take, we have a corresponding probability density. A third dimension of interest is risk. Although risk is not defined in mathematical terms in a generally accepted form, we can say that risk is a measure of how big the chances are that an action affects us adversely. Therefore we think that risk can be interpreted as second order information that can itself be expressed in terms of a property (the level of risk) and a probability density function over the range of possible risk levels.

Since we did not find a decision support system or even a tool-kit that is capable of dealing with these dimensions elegantly and consistently, we decided to design such a system and we will present the results in the following.

3 Design of DeTerminator

3.1 Overview of Decision Support System

Trying to determine the scope of our project, a decision support system, we want to shoot at the following problem areas: we want to preserve the freedom to have information available in a wide range of formats. Ideally, we would like to keep the option open of supplying the system with information that is extracted from arbitrary sources, including text analysis systems, laboratory equipment, commercially available data streams (ticker tape) or output from other programs. Since we want a general decision support system - rather than a solution to a given problem, we cannot rely on fixed algorithms such as linear programming or network flow, but have to be able to express expert knowledge that exists in a multitude of forms. Our system must capture the semantic dimensions of risk, probability and properties. Since we do not know whether the action that is based upon the output of the decision support system is performed by man or machine, our system should provide information in a form that is suitable for treatment by both of them.

More specifically, we want to be able to treat data properties with a process of elimination, e.g. if we have a property *color* that can range over the whole spectrum from red to violet, we should be able to exclude certain colors as possible while all the rest are still being considered - although to varying degrees. We call this dealing with multivalent data.

Since the traditional field of application for decision support systems can be found in customized applications that are designed for specific purposes rather than off-the-shelf solutions, it seems imperative to us that the tools used to construct such decision support systems should empower less specialized people than the original authors of those tools to create a system. We thus want to create a user-friendly, graphics-oriented tool-set.

Our system determines many properties and variables in order to exterminate the decision problems at hand; hence, we call our system *DeTerminator.*

3.2 Choice of Expert System

Given the tool-set approach decided upon above, an expert system-like tool is a plausible choice.

Given the other requirements stated before, it seems a natural consequence to evaluate various multi-paradigm expert systems such as Babylon "that claim to be independent of the various domains by supporting various representation formalisms. [...] These systems are referred to as hybrid systems as they provide the possibility to alternate between or to combine various knowledge representation formalisms in the development of knowledge bases." [Babylon 1989]

For various practical reasons such as general availability, access to source code and cost, we decided to look more closely at Babylon. During our evaluation process, we encountered several severe deficiencies in the paradigms available in Babylon as well as in other paradigms that are generally used to solve problems of the type we are looking at. This ultimately forced us to think of a new paradigm for representing expert knowledge. The remaining choice was between enhancing Babylon by adding yet another paradigm or to create a system solely based upon our new approach. In the end, we decided for the latter for two reasons: first, we wanted to show that the method was powerful enough to stand on its own and second, for technical reasons, such as execution speed, it seemed more reasonable not to base our system on a relatively big symbolic computation environment.

3.2.1 Paradigms in review

In the following, we show what particular weaknesses and problems arise with the standard paradigms used in many decision support systems, and we will lead to an escape from the limitations in expressive power imposed by the standard paradigms.

Constraints

Constraints restrict the possible values of an attribute, i.e. the set of possible values that is defined for this attribute. In mathematical terms, we can define this restriction as reducing a defined domain to a particular subset. [babylon 2.3.4] Consequently, constraints can be combined to form systems with no, one or several solutions; thus, we can say they support multivalence.

Constraints are useful for second order knowledge, e.g. to verify boundary conditions.

It is awkward, however, to explicitly state facts. Also, constraints do not support an integrated way of dealing with likelihoods.

Frames and Scripts

Frames share the advantages of object-oriented programming with those of symbolic computation. Their object-oriented nature endows them with the powerful capabilities of inheritance, meaningful grouping of data and operations, and an intuitive way of expressing behaviors.

On the downside, frames and scripts resemble more an implementation language than a declarative form of knowledge representation. They don't directly support the semantic dimensions we require to express and process our knowledge in. Of course, frames and scripts could be used as an implementation language for a paradigm that supports these dimensions.

Rules

Rules are a nice, declarative way of expressing a knowledge base separated from the underlying evaluation mechanism. It separates the process of knowledge engineering from the programming aspect of a decision support system and is therefore suitable for the construction of a variety of "mission-critical"[1] custom decision support systems. Differently said, an evaluation program can be delivered as an off-the-shelf solution while the necessary knowledge is filled in by unrelated knowledge engineers.

Rules do not, however, easily allow sufficiently fine grained levels of expression on a symbolic level which is where rules shine. To express properties that can have subtle, but important differences such as the gray level of an area in a satellite picture, there is virtually no other means than to take refuge to numerical representations.

Bayesian Inference

Bayesian Inference maintains a coherent, mathematically sound framework to express likelihood in terms of probabilities. Given correct information on the dependence of the data, Bayesian inference guarantees correct results.

Bayesian Inference, however, does not attempt to solve the problem of representing multivalent data.

---

1. "For businesses to compete in the 1990s, requires effective management of one of their most valuable assets, their information. More than ever, organizations are realizing that [...] accurate and timely information is a strategic competitive weapon and vital to organizational success. But as an organization's information requirements increase, MIS departments face a growing backlog of applications. And, often a bottleneck occurs as they try to provide decision-makers with what they desperately need - the right data, on time, presented in an intuitive fashion." [NeXT 1992] The applications designed to fill these bottlenecks are called mission-critical custom applications.

Probability Function Semantics Evaluation (ProFuSE)

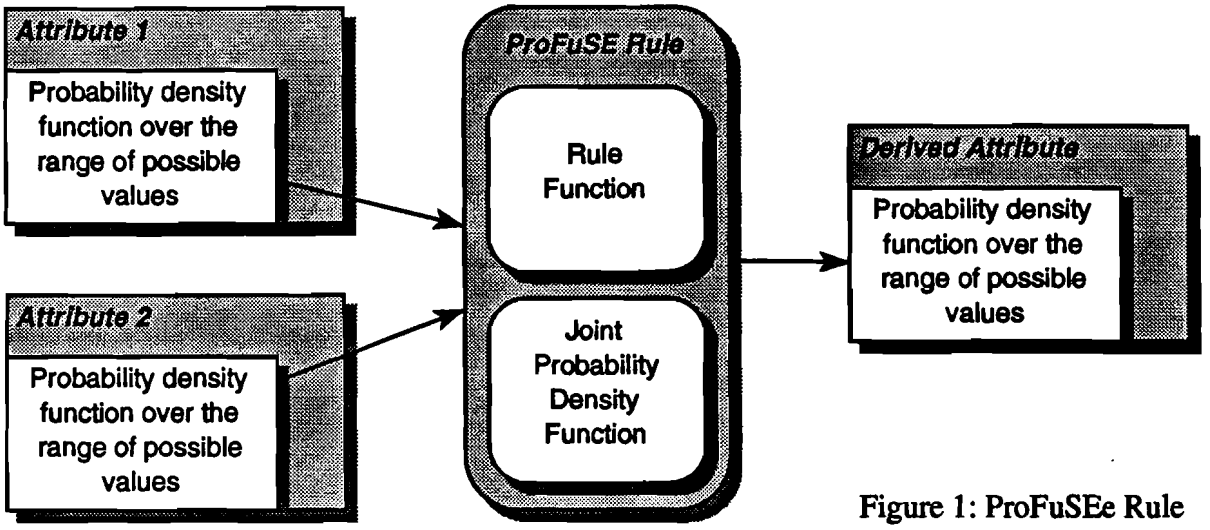


Figure 1: ProFuSEe Rule

After we have seen the various difficulties that other schemes confront us with in the effort to express multivalent data in conjunction with a mathematically sound notion of likelihood, we decided to represent attributes in our system neither by some symbol, nor by a mere number, but rather with a function. The approach we take is to map the entire domain over which a property is defined into a range of real numbers and by defining a probability density function over said range, we can associate a probability with each possible value of the attribute in question.

By employing a combination of rule functions and joint probability density functions, we can soundly derive other attributes from known attributes. How this is done will be explained in more detail later.

We believe that this approach constitutes a means of combining the positive aspects of Bayesian inference and rule based systems, and incorporates support for multivalent data. Also, to some degree, we can incorporate constraints explicitly into our rule functions.

An implementation based on object-oriented tools should allow us to group data (and operations on them) in a meaningful way as well as to elegantly interface them with other programs, user-interfaces, etc.

To get back to the more fundamental properties of this form of attribute representation, we can see that we managed to achieve our goal to express three semantic dimensions of particular importance to decision support systems: property, probability and risk.

We preserve the property dimension because we map all possible values a property can assume to a range of rational numbers. Thus, since we choose a unique mapping, it can be reversed and the property values can be retrieved.

Since for each possible value, there exists an associated probability density value, we can compute a probability for an arbitrarily narrow range of possible attribute values (by integrating the probability densities over that range).

The concept of risk is not uniquely defined. Relatively spoken, it confronts us with the hardest problem: there are schools of thought that define risk as a function of probability; others, however, consider risk to be a property that is completely orthogonal to probability, i.e. the probability of an event happening and the risk associated with a decision that depends on this event are not a direct function of each other. We believe that the information provided by the property and probability

dimensions is sufficient to allow for a second order ProFuSE rule network to extract risk if it is defined as an orthogonal property.

3.2.2 ProFuSE

Now that we have shown how we can escape from the limitations in expressive power and have a system that is capable of representing properties, probability and risk, we take a look at the methods employed to do so and investigate how we can approximate certain scenarios of knowledge representation such as Bayesian inference, Boolean expressions and linear dependencies.

3.2.2.1 Mathematical Interpretation of the ProFuSE Method

Our aim is to arrive at a probability density function over a mapping of a domain of possible attribute values onto a range of real numbers starting from several (in our case two) other attributes of which we already have a valid representation of the same form. To achieve this, we need a rule function and a joint probability density function.

The first is a function that reflects a relationship between any possible value of all the input attributes and the corresponding value for the attribute to be computed.

The second is a function that computes the joint probability density for all possible combinations of mapped attribute values of all the input functions given their respective probability densities. In the case of independent inputs, this function simply corresponds to a multiplication of the probability densities.

The remaining task is to obtain a mapping from the rule function and the joint probability density function to the probability density function of the mapped attribute values of the computed attribute. We know that every combination of mapped input attribute values that results in the same computed attribute value has the same output of the rule function. Thus, we can obtain a probability density for any mapped value of the computed attribute by integrating the joint probability density function over the inverse image of the computed attribute under the rule function. Graphically spoken, this corresponds, in the continuous case, to the integration of the joint probability density function along the contour lines of the output of the rule function.

In mathematical terms, let $p(x), p(y)$ be the probability density functions on the mapped values of the computed attribute; they have a joint probability density function $j(x, y) \equiv p(x, y)$. Our rule function is $z = f(x, y)$. Given those, the probability density function of the output of the rule is

$$p(z) = \sum_{(x,y) \in f^{-1}(z)} j(x, y)$$

where $f^{-1}(z)$ is the inverse function of the rule function. See illustration on figure 2.

3.2.2.2 Approximation of Standard Paradigms

Bayesian Inference

ProFuSE is equivalent to Bayesian inference, but is concerned with ranges and probability density functions over such ranges rather than with probabilities for certain fixed attribute values.

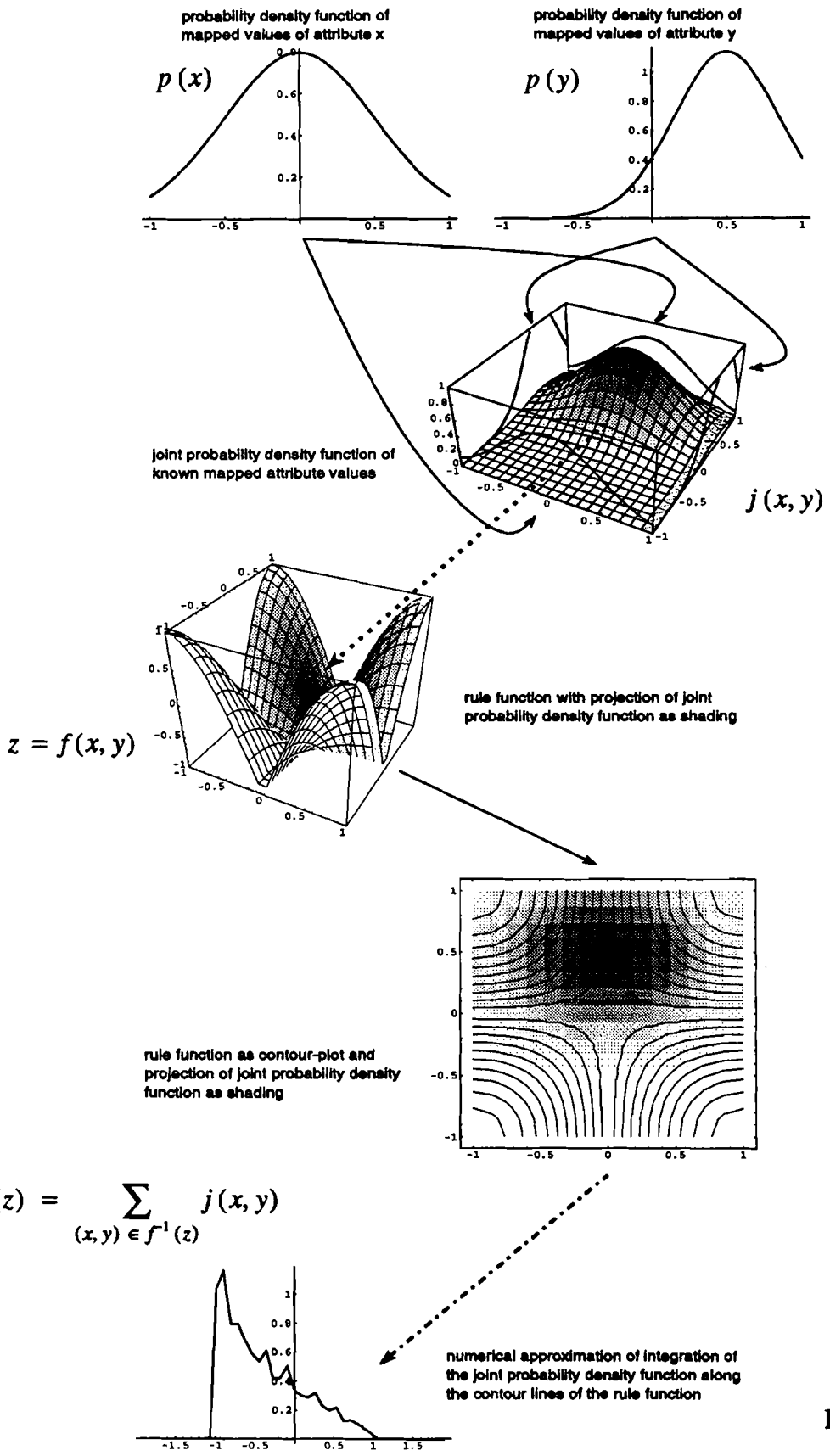

probability density function of mapped values of attribute x
p (x)
probability density function of mapped values of attribute y
p (y)
joint probability density function of known mapped attribute values
j (x, y)
z = f(x, y)
rule function with projection of joint probability density function as shading
rule function as contour-plot and projection of joint probability density function as shading
p (z) = Σ j (x, y)
(x, y) ∈ f⁻¹ (z)
numerical approximation of integration of the joint probability density function along the contour lines of the rule function

Figure 2

Boolean Expressions

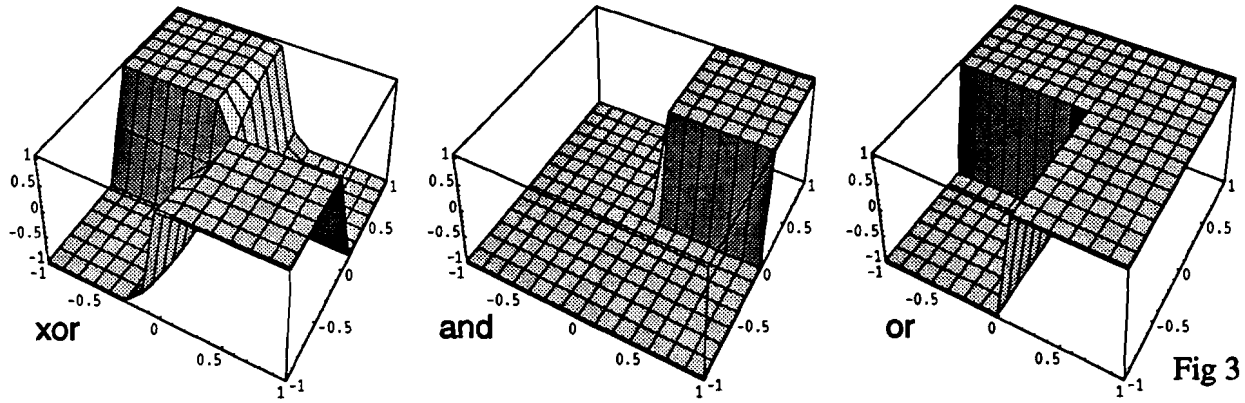


Fig 3

By choosing appropriate rule functions, our system can relatively easily approximate rules based on Boolean logic. The way this works is similar to the implementation of logic circuits in digital electronics where logic values are mapped onto voltages or currents, which - of course - are analog quantities. By using functions on real numbers as the ones depicted above, we can emulate Boolean functions as long as we adhere to similar restrictions as those found in digital electronics, i.e. we cannot use the whole mapped range, but rather have to define intervals that we consider a logical "1", others that we consider a logical "0" and a everything in between that we consider invalid. In our case, for example, we could use values below -0.5 as logical "0" and values above 0.5 as logical "1". Values between -0.5 and +0.5 are considered invalid.

Nothing stops us, however, to have true Boolean expressions encoded in functions. If one wanted to have a learning system integrated to the rules, though, one might want to choose an approximation of the Boolean function using a continuous, differentiable function such as the one displayed above for *xor*. More about learning in the outlook.

Linear Dependencies

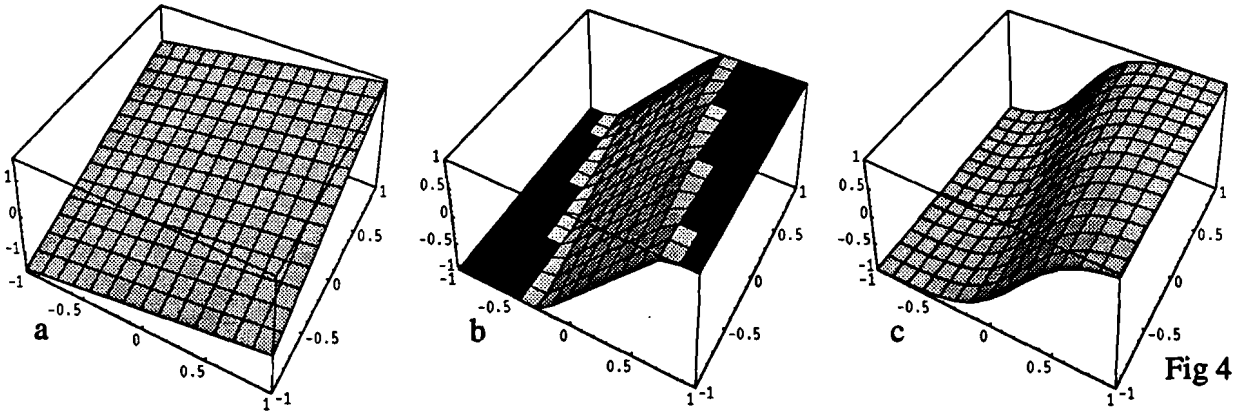


Fig 4

In many cases, approximation of linear dependencies is a straightforward process: this is the case when the range to which the function maps results in a derived attribute that can again be used as a ProFuSE rule input. To simplify matters, however, and to be able to arbitrarily combine derived attributes, the range of possible values should be unified, i.e. in graph *a* above, the inputs as well as the output range from -1 to 1. Graph *b* also depicts a linear dependency, but the output as denoted by the black areas on the graph exceeds the range [-1,+1]. If the vast majority of the data lies within the defined (gray) range of the function, we can use a squashing function as a rule

function that corresponds as closely as possible to the linear mapping where the linear mapping is defined, but is guaranteed to yield results within the defined range for all cases (graph *c*).[1] Alternatively, one could scale the linear result down to defined range. The second approach is appropriate when the data samples are more uniformly distributed over the defined range.

3.2.2.3 Custom Functions

Just as we can express Boolean functions, any discrete set of values can be mapped in such a way that it is feasible to be used as an input attribute for a ProFuSE rule. We believe that a well-chosen combination of rule functions and attributes will allow us to express a great variety of transformations relatively easily.

3.2.2.4 Symbolic versus Numerical Evaluation

Symbolic integration in general, and along contour lines in particular, is, in all but the most trivial cases, a very difficult and often even impossible task. For that reason, we chose to use a numerical approximation to arrive at our solutions. In order to trade off speed versus precision, our implementation allows us to divide the range into an arbitrary number of equally sized intervals.

Whenever the slope of the probability density function is relatively steep and the division of the range uses relatively large intervals, there are problems in smoothness of the resulting output. In cases where this causes problems, decreasing the interval size can help in solving the problem.

3.2.2.5 Notes on Computational Complexity

In the following, we want to address some considerations on the computational complexity of the numerical approximation discussed above.

Single Rule Update

$$R = O\left((J+F)\prod_{i=1}^{\#\text{ of inputs}} \#\text{ of intervals}_i\right)$$

The update-time for a single rule is shown by the formula above with J being a bound on the time complexity of the computation of the joint probability density function j, and F corresponding to a bound on to the time complexity of the rule function f.

Network Update

There are two modes in which our system can operate: in one mode, we assume that for each desired output, we have a complete set of inputs that change simultaneously and cause a complete update of the network. We shall call this off-line operation. Alternatively, we allow any input to change at any time and have the whole network be updated appropriately to reflect the new situation. We call this on-line operation.

According to these differences in operation, we can influence the evaluation strategy of the network to make computation more efficient.

1. The squashing function used is a three-dimensional extension to the sigmoid function with the output range adjusted to map to [-1,+1].

Typical Case

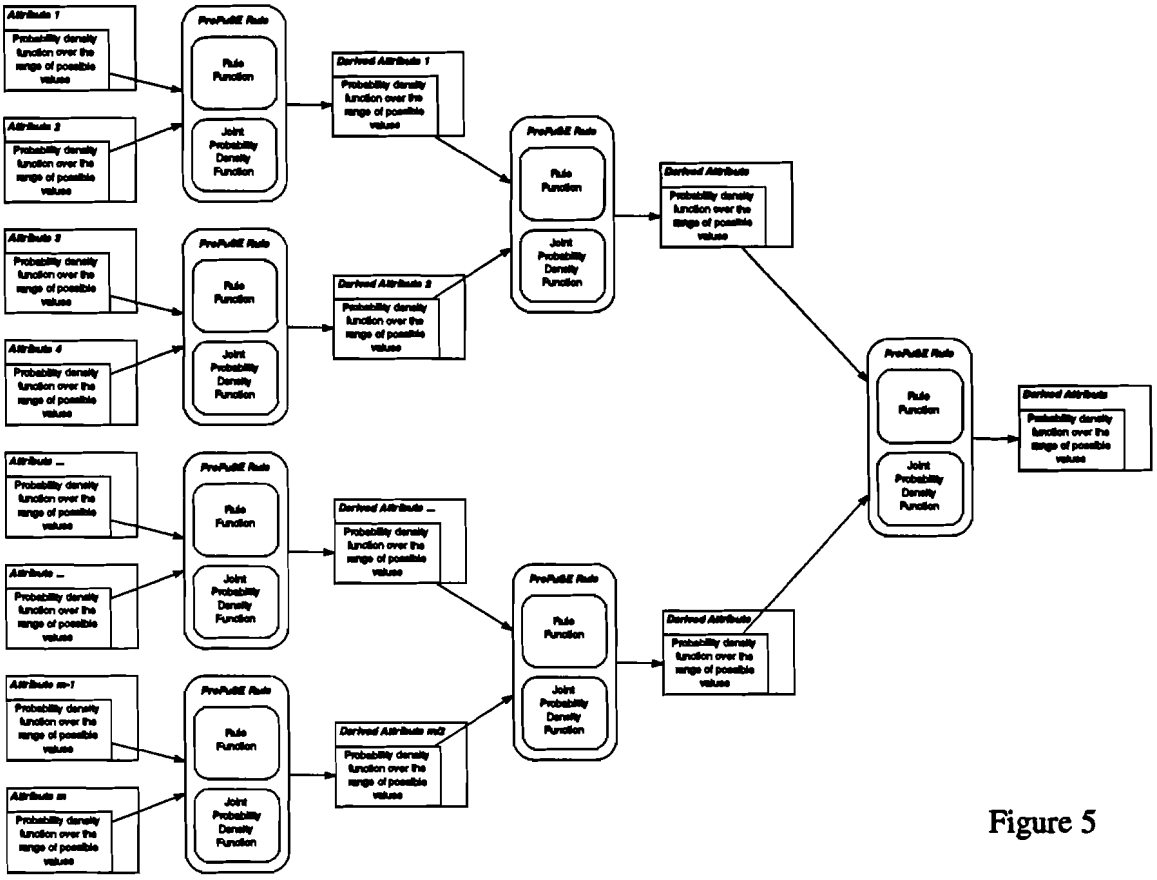


Figure 5

As a typical case, we consider a balanced, binary tree network such as the one shown above. Taking m as the number of inputs, n as the number of rules, thus $n=m-1$, and R as the update time for a rule, we achieve the following upper bounds:

In off-line mode, a complete update of the net as well as a change in a single input requires time

$$O(Rn)$$

This result holds because before each complete network update, all inputs to any given rule are declared invalid and thus during the update, a result is only propagated to the next level after all inputs to the rule have been updated (and thus validated).

In on-line mode, a single input change requires time

$$O(R\log n)$$

because at most one path from one input to the root of the tree is affected. Therefore, the update time is proportional to the depth of the tree. A complete update in on-line mode is thus

$$O(mR\log n)$$

since for all m inputs, an update has to be performed.

Worst Case for 2 inputs

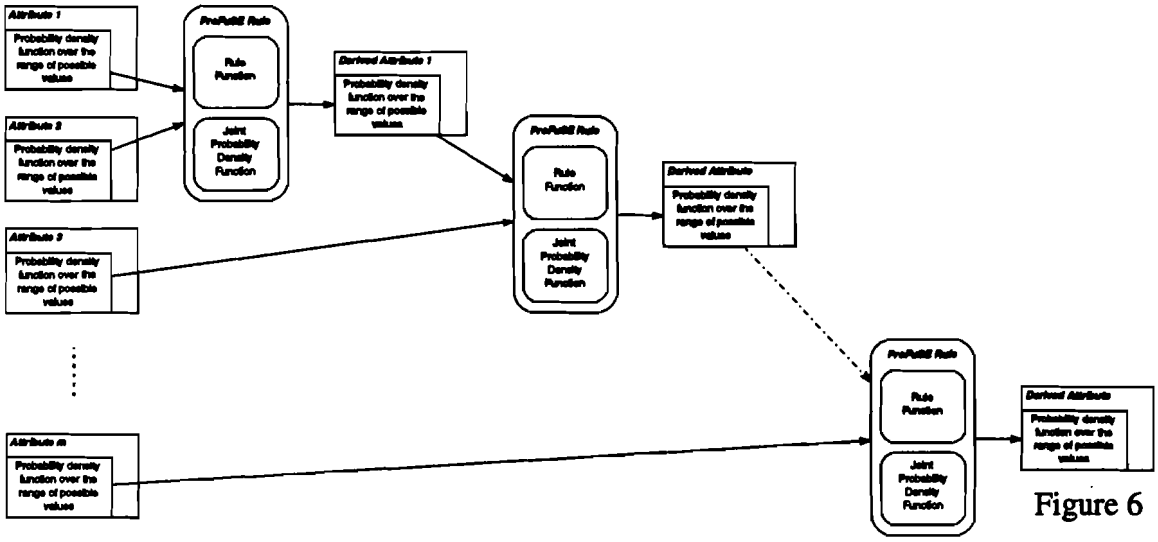


Figure 6

In off-line mode, worst case and typical case are the same.

In on-line mode, the worst case of a single attribute update requires time

$$O(Rn)$$

since the attribute at the deepest level of the tree affects all rules. A complete update in on-line mode is in the order of:

$$O(Rn + Rn + R(n-1) + R(n-2) + \ldots + R) =$$

$$O(R(n + \frac{n(n+1)}{2})) =$$

$$O(Rn^2)$$

3.2.2.6 Isolation of Error Sources

A particularly positive property of our ProFuSE rules is that they isolate the various error sources in a system dealing with probabilities as well as with attributes. Since our rule functions and our joint probability density functions are separated, we can more easily lock one of these while examining the other one. This is especially useful in cases where one of them is known to be correct.

4 Implementation of DeTerminator

4.1 Implementation Language

We decided to use Objective-C as our implementation language. This was influenced by several factors:

- execution speed of numerical computations
- object-oriented programming language with
  - dynamic binding
  - improved polymorphism (in comparison to C++)
- native language on the NeXT computer, on which the program has been implemented
  - extensive application programmer interface (API)
  - Interface Builder support

4.2 Notes on Implementation Structure

4.2.1 Program Structure

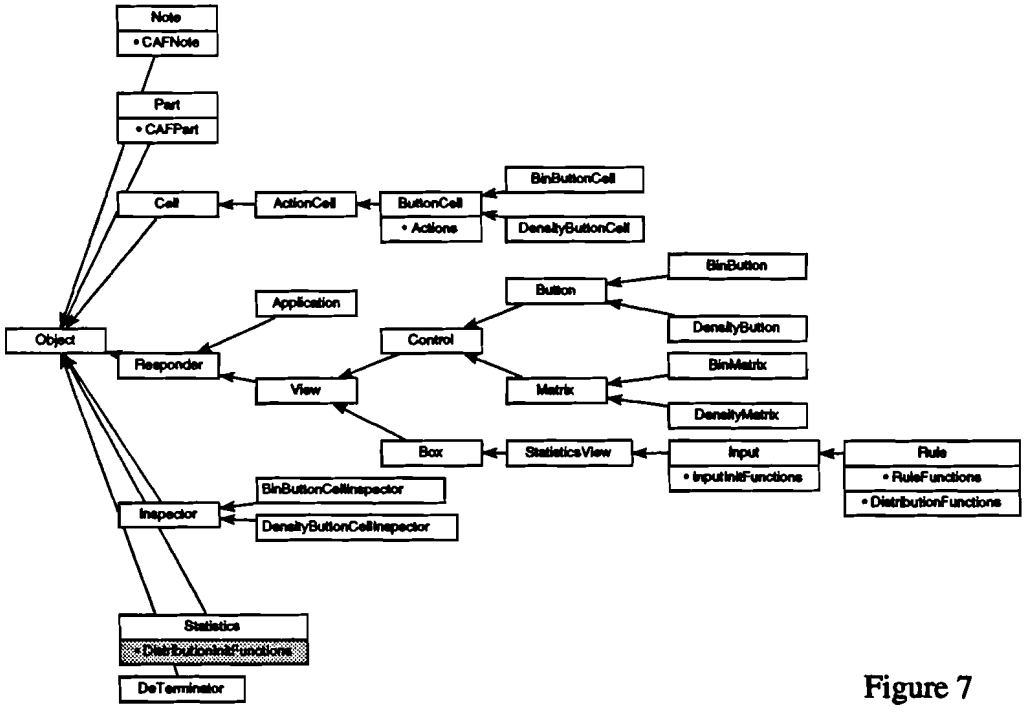


Figure 7

The program heavily builds upon the class hierarchy shown above (in blocks with multiple fields, the lower fields correspond to class categories, i.e. code extensions to a previously defined object). Our program is event-driven and has a graphical user interface: the knowledge engineer creates a custom-decision support system primarily from building blocks representing inputs and rules. The knowledge engineer decides on a List[1] of Inputs, a set of Rules, and how they relate to each other.

1. In the following, we use the Helvetica font to indicate words that directly correspond to class names.

For an Input to be well defined, it needs an *Input Initialization Function (IIF)* and a *Distribution Initialization Function (DIF)*. To integrate an Input into a network, it also needs to have information on what Rules are immediately dependent on this Input.

The knowledge engineer is given Input instances of the type shown that are provided by a customized version of the NeXT InterfaceBuilder that was developed for this project by the authors. He[1] specifies the methods used for the IIF and DIF. The dependencies are expressed by connecting the output of the Input with the inputs of the Rules immediately dependent on them; such a network is illustrated further below. When the knowledge engineer has created the network, the user will see the same instances; he cannot change the network structure, however, since the particular setup has been compiled into an executable. He can, though, change the IIF and DIF at run-time if the knowledge engineer enables that functionality. As indicated by the windows captured off the screen, one can choose between a display of the probability density function and the distribution function for the output of the Input. Furthermore, the number of intervals over the defined range can be modified for all Inputs individually, at run-time.

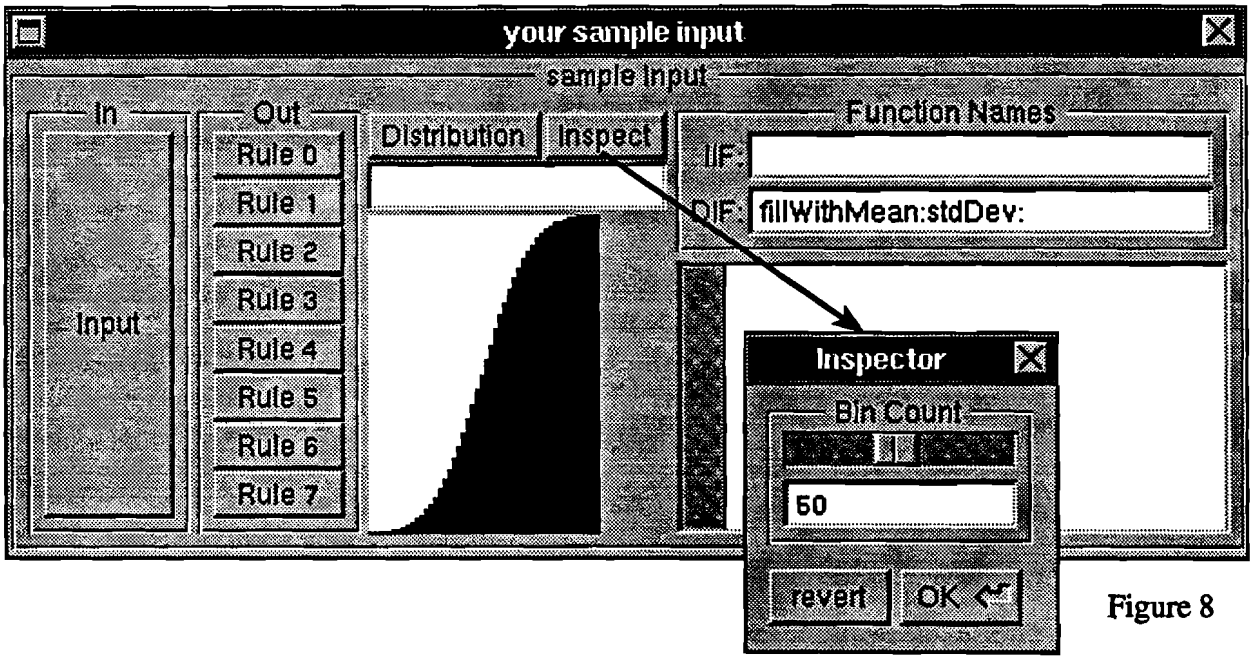


Figure 8

Similarly, the knowledge engineer specifies a *Rule Function (RF)* and a *Joint probability Density Function (JDF)* for each Rule. The Rule class inherits most of its functionality from the Input class. A Rule, however, may have more than one Input it is linked to; generally, two Inputs together with the RF and the JDF define the output of the Rule. Again, the knowledge engineer specifies other Rules immediately dependent upon on the current one. If no dependent Rules are specified, the Rule is considered to be a root node of the network. Note that a network may have more than one root node and thus allows for an output with more than one attribute.

Using this graphical approach and by not differentiating between a Rule within the network and the root Rule(s), we are able to examine any rule when the system is running. This should encourage the knowledge engineer to choose meaningful rules so that intermediate results can be understood by the user who might want to inquire how the decision support system derived its result.

1. For the sake of political correctness, he refers to a man, where man is a generic term for human being (→ mankind).

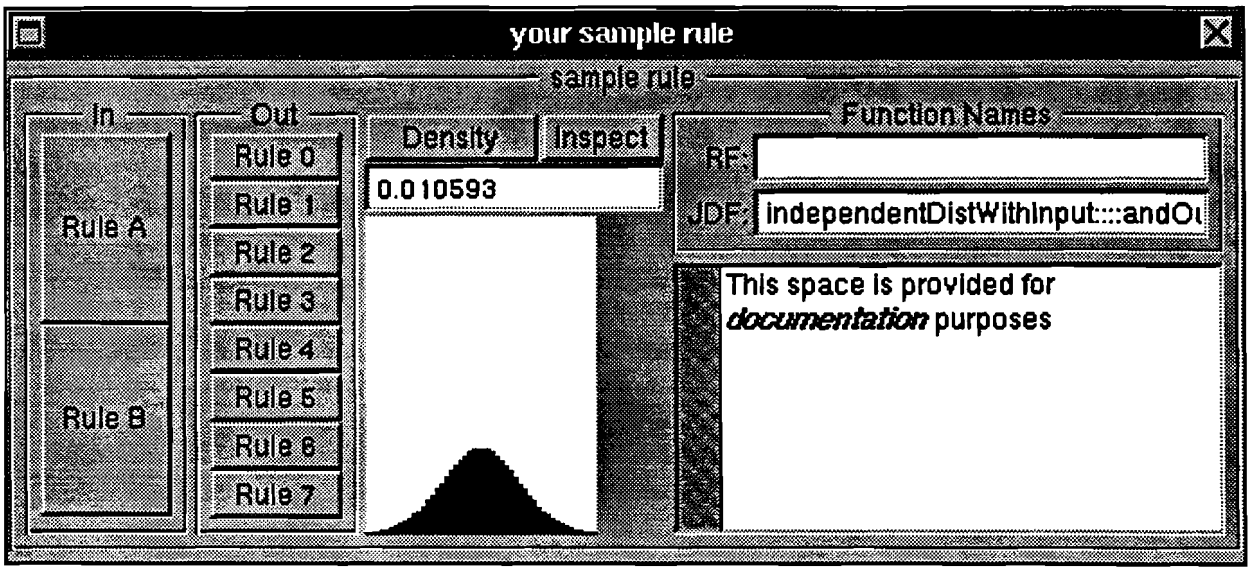


Figure 9

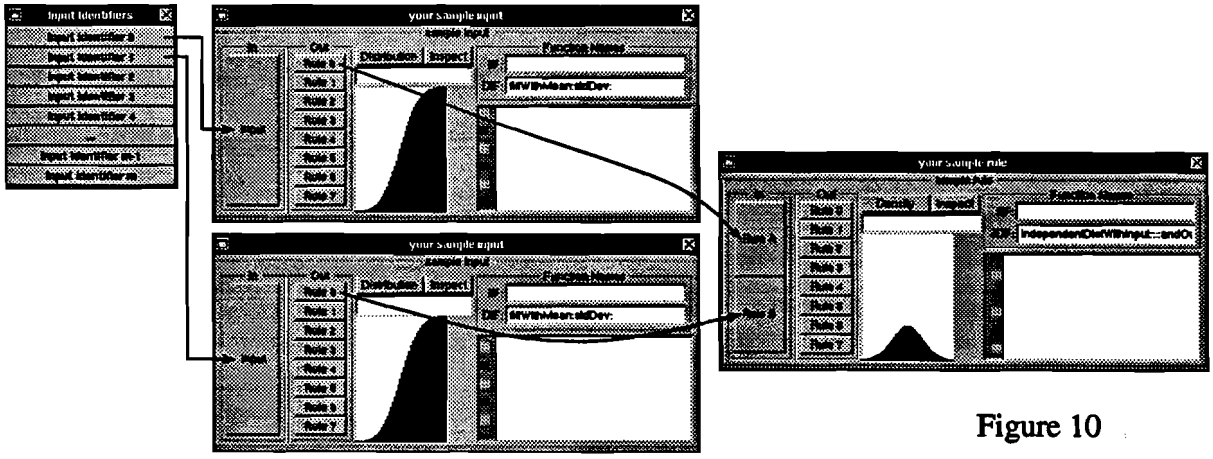


Figure 10

4.2.2 Data Structures

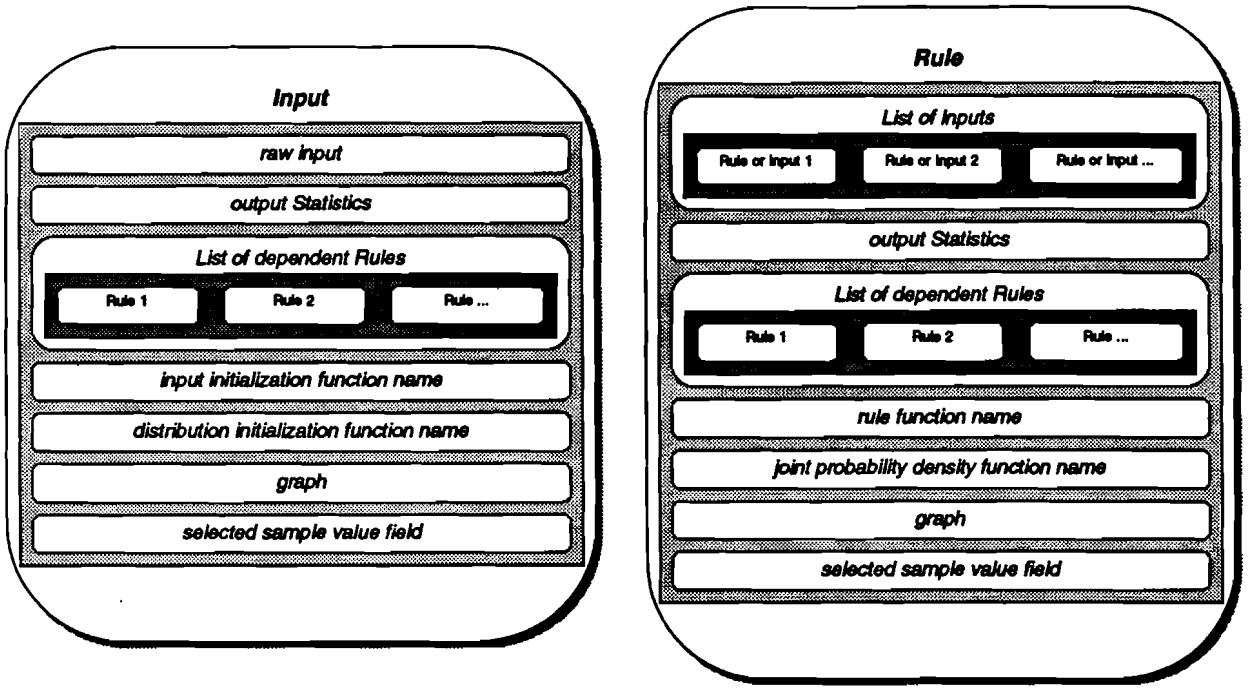


Figure 11

In the picture shown above, we can see the structures of our main building blocks, the Input and a Rule object. In the diagrams, a gray rectangle contains the instance variables of the object it is drawn in. It symbolizes the data abstraction and the protection of these instance variables (if there were any public instance variables, they would be shown outside the gray rectangles). Of course, objects can contain other objects that again have instance variables as can be seen above, e.g. with the List of Inputs.

The other data structures we use only have supporting functionality, and we therefore refer the interested reader to the source code.

4.2.3 Algorithms

In the following, we will explain how the core algorithms of our system work.

Propagation of changes

In the case of on-line operation, we recalculate our output for any change in the List of Inputs, and then, if such an output change is significant, send a notification of a change in our output to every rule in the List of dependent Rules. It is up to the knowledge engineer to decide what constitutes a significant change.

```
if(some input has changed)
then
      calculateChange(changed Input)
      if(change signficant)
      then
            for(each dependent rule)
                  update(dependentRule)
```

In off-line mode, changes are only calculated after all Inputs to a Rule have changed since the last update. Thus, the pseudo-code looks as follows:

```
if(some input has changed)
then
      if(not(there is some unchanged input))
      then
            calculateChange(changed Input)
            for(each dependent rule)
                  update(dependentRule)
```

Calculation of changes

Below, we show the pseudo-code of the routine computing the probability densities corresponding to the output of a rule function. Note that this pseudo-code assumes that there are only two inputs (x and y). In the case of n inputs, we need a n-deep nested arrangement of *for*-loops or a comparable, recursive iteration construct.

```
zero(outputPDF)
for(all x)
      for(all y)
            ruleFunctionResult:=ruleFunction(x, y)
            jointProbabilityDensity:=jointProbDensityFunction(x, y)
            outputPDF[ruleFunctionResult]+=
                  jointProbabilityDensity
                  * binSize_x
                  * binSize_y
                  / binSize_outputPDF
```

5 Empirical Test of Modelling Tool

5.1 Application Domain

Due to our previous exposure to economics and in particular to finance, it was clear to us that we wanted to test our system on a problem of the finance domain. Although there are many interesting aspects that could be tackled in that domain, such as financial valuation, bond ratings, portfolio management, arbitrage, market prediction, trading and news understanding, we had to restrict ourselves due to limitations in time, available information and our own knowledge.

5.1.1 Constraints due to Information Available

The difficulty in obtaining reasonable amounts of consistent fundamental data such as earnings reports, quarter and annual reports, etc., excluded from the beginning any kind of expert system working with this data. Fortunately, we had access to general market information such as the *Dow Jones Industrials Average (DJIA)* and the volume of the *New York Stock Exchange (NYSE).* These kind of data, called technical data, and numbers derived from those, so-called technical indicators, is what our system is based upon. It is known that if the *Efficient Market Hypothesis* holds true, prices of securities must be good indicators of value, i.e. prices "fully reflect" available information and do not have predictive power [Fama 1976]. However, other approaches, such as Mandelbrot's *Stable Paretian Hypothesis* [Mandelbrot 1964] suggest that the distributions underlying financial market time series are of the class called *Stable-Paretian*[1]. The *Fractal Market Hypothesis* [Peters 1991] says the set of parameters to the Stable-Paretian distribution is such that the resulting time series have a fractal characteristics and therefore are statistically self-similar with respect to time. This violates the independence assumption underlying the Efficient Market Hypothesis, invalidates the arguments technical analysis of financial markets and therefore allows us to consider an approach based on technical data. Nevertheless, we do not claim that our system will successfully predict market behavior, especially since the choice of indicators was severely limited by the constraints we had in respect to time and information access.

5.1.2 Constraints due to Know How

Our project is solely based on published methods for technical indicators. Moreover, most of these indicators have been designed to be used on their own rather than in combination with other indicators. Our approach, however, tries combines these indicates to create an overall picture of the market. Due to restricted resources, we could not undertake extensive research on what indicators are most suitable to be used in such a combination and what kind of combination produces the best results. We therefore have to assume that the performance of our system is suboptimal. Nonetheless, since our goal is to show the feasibility of such a system and not to find the perfect knowledge-base, the above mentioned constraints and restrictions did not stop us from pursuing this project.

1. Also called *Pareto* or *Pareto-Levy*

5.1.3 Application

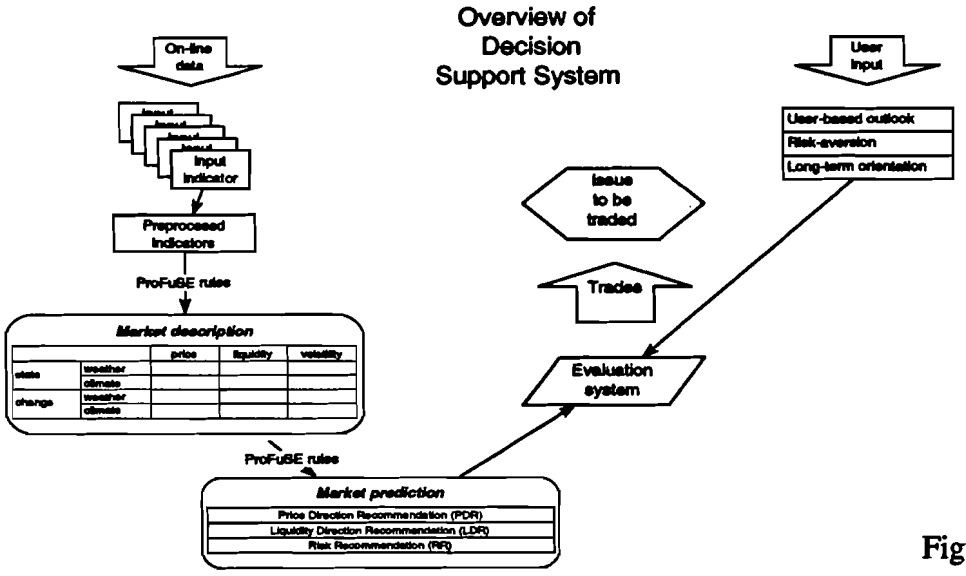


Figure 12

We decided to create a system that could possibly form the core of an automated trading environment in such a way that it receives certain preferences from a user as well as a constant data stream representing the public knowledge of the market. The system is supposed to arrive at its output in two steps: In the first step, it tries to derive a picture of the current market situation, and, in a second step, it recommends plausible reactions based upon this picture of the market situation.

5.2 Modelling the Knowledge Base

5.2.1 Requirements

Following the two-step approach mentioned before, our first effort is to find a tenable representation for a particular market situation as well as a representation for action recommendations. The second step is to come up with a rule base that is capable of producing the results necessary to assign meaningful values to these representations.

Representations of Market Situation and Action Recommendations

• *Market Situation*

Analyzing the market situation, we find that there are several dimensions of interest: the first is a set of properties that define a particular point in the time series. The second is that if we consider not only a point, but the time series as a function, we have a value and a trend that is applicable to each property in the set. As the third dimension, we define how the combinations of the previous aspects vary depending on the scale at this time series is examined. The following table represents all valid combinations of the second and third dimension mentioned.

	state	change
weather	The immediate outlook, the current condition	The current trend; a short-term moving average's first derivative
climate	Value of a longer-term moving average	The longer-term trend; a long-term moving average's first derivative

It remains to determine the set of properties that define the first dimension. Besides price, we deem it valuable to include liquidity and volatility in our considerations. The choice of price is obvious, since it is the target around which the market circles. Liquidity is important because it determines the ability of a potential market participant to engage in trading. Lastly, volatility is worth consideration since it tells us something about market fluctuations and therefore about our ability to obtain a price that is as close as possible to the last recorded price on the same market issue. Consequently, we receive the following table that we have to fill the following slots:

		price	liquidity	volatility
state	weather			
	climate			
change	weather			
	climate			

Note that in our system, each slot in the table corresponds to a probability density function over the range of all the values that the respective slot can take.[1] Since there are no well-defined units in which to express notions like liquidity and volatility, and since price has a trend, and therefore has no meaning when used decoupled from the date, we map the values of all the slots in a range that is defined between *LOW* and *HIGH*.

• *Action Recommendation*

In order to act in financial markets, there are three notions that are generally considered of importance: the first and most obvious one is the question in which direction the market price will develop. The second one is whether the liquidity in a market increases or decreases since depending on the level of liquidity, certain transactions are impossible or influence the price at which they can be executed in a negative way. An idea of the liquidity to be expected in a particular market is therefore crucial in determining which financial instruments can be successfully employed (i.e. the choice between equity, option and/or futures trading is influenced by the liquidity). Lastly, we consider risk. Risk, although not precisely defined in general, is spoken about almost constantly when financial transactions are undertaken. In our system, risk is a second order measure that tries to capture on the one hand how well different parts of the system agree in their assessment of the current situation, and on the other hand it tries to capture trends in volatility and liquidity that might handicap the trader's ability to perform actions of his choice in the market. The slots representing these three measures are called *Price Direction Recommendation (PDR)*, *Liquidity Direction Recommendation (LDR)* and *Risk Recommendation (RR)*. Again, these measures are considered relative and therefore mapped into the range *LOW* to *HIGH*.

5.2.2 Notes on the Knowledge Base

To give some idea what goes into the system that we use to test our overall approach, we comment in the subsequent sections on some of the indicators, the rule functions and joint probability density functions used.

5.2.2.1 Indicators

Since most technical indicators, in their basic form, only have a fixed value for any given point in time rather than a distribution, we decided that we interpret this value as the mean, and the

1. Each slot corresponds to a property in terms the ProFuSE method. We avoid the word property at this place to prevent confusion with the set of properties mentioned as a dimension in the table shown.

accuracy and reliability of the underlying data source as the standard deviation of a normal distribution that is generated from the raw input indicator to serve as an Input to our system.

A Empirical Trading Rules

In contrast to indicators, for which there is mostly some reasoning why they should work (although they are usually crude approximations), empirical trading rules are *"truths"* that can be observed, but the insight into the reasons of their existence is usually rather obscure. Furthermore, it is quite often difficult to express them in precise mathematical terms. A couple of examples follow - they are primarily discussed in Yale Hirsch's *Don't Sell Stocks On Monday* [Hirsch 1986]:

• *Don't Sell Stocks On Monday*

While the market from 1952 to 1984 was gaining 930 Dow points, Mondays alone were losing 1565 points. 43.1% of Mondays close higher than the previous trading day on average. Even during bull years, only 47.7% of Mondays closed higher, whereas in bear years, 35.1% closed higher.

The so-called Monday effect is often contributed to the preference of companies to release bad news on weekends in order not to cause a market panic. Although this is a plausible explanation, at a closer look this argument is a little bit shaky because it would attribute bad news in a few companies with overall market down-movement.

Although the Monday effect is not strong enough to allow profitable trading on its own (consider commission cost), it might give us a hint on when not to realize profits of longer term investments.

• *Election and Pre-Election Year*

It is no mere coincidence that the last two years of the 38 administrations since 1832 produced a total market gain of 515%, dwarfing the 8% gain of the first two years of these administrations.

These results can be explained by what economists call the Nordhaus-political-cycle [Nordhaus 1989]: this theory says that in order to be re-elected, the government has to stimulate the economy and lower unemployment rates in pre-election years since the short memory of average citizens only looks at the most recent performance of a government, whereas government spends the rest of the time stabilizing the economy, lowering inflation rates, and therefore also producing unemployment. This leads to an overall healthier record of the administration that is considered by the more long-term oriented leaders of the economy and sponsors of presidential campaigns.

• *The Santa Claus Rally*

"Santa Claus comes to Wall Street almost every year with a short, sweet respectable rally. In the past thirty-three years he has appeared twenty-six times. The rally occurs within the last five days of the year and the first two in January. When Santa doesn't call, beware. Five of the years he failed to show up preceded bear markets."[1]

There are various opinions on why and whether the market should go up the first few days of the year. It is a common opinion that the first few trading days of a new year are a good barometer for the entire year (that's a different rule, though).

B Technical Indicators

Many of the technical indicators discussed below are described in [Colby 1988].

• *Moving Average Convergence-Divergence (MACD)*[2]

A shorter term moving average will rise more quickly than a longer term moving average during market up-trends. As the rise comes to an end, the slower moving average will catch up, narrow-

1. Quoted literally from *Don't Sell Stocks on Monday* because of the nice wording
2. invented by Gerald Appel, Signalert Corporation, New York

ing the distance between them. This narrowing suggests an end to the advance. The same pattern occurs during market down-trends.

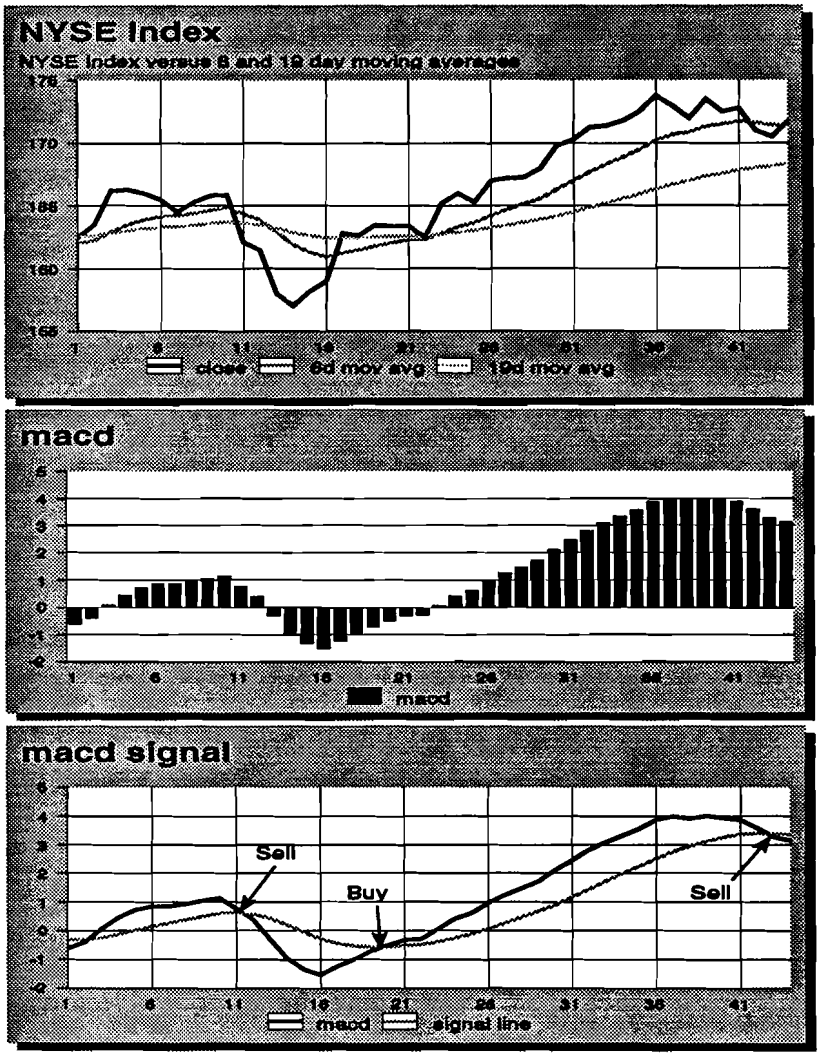


Figure 13

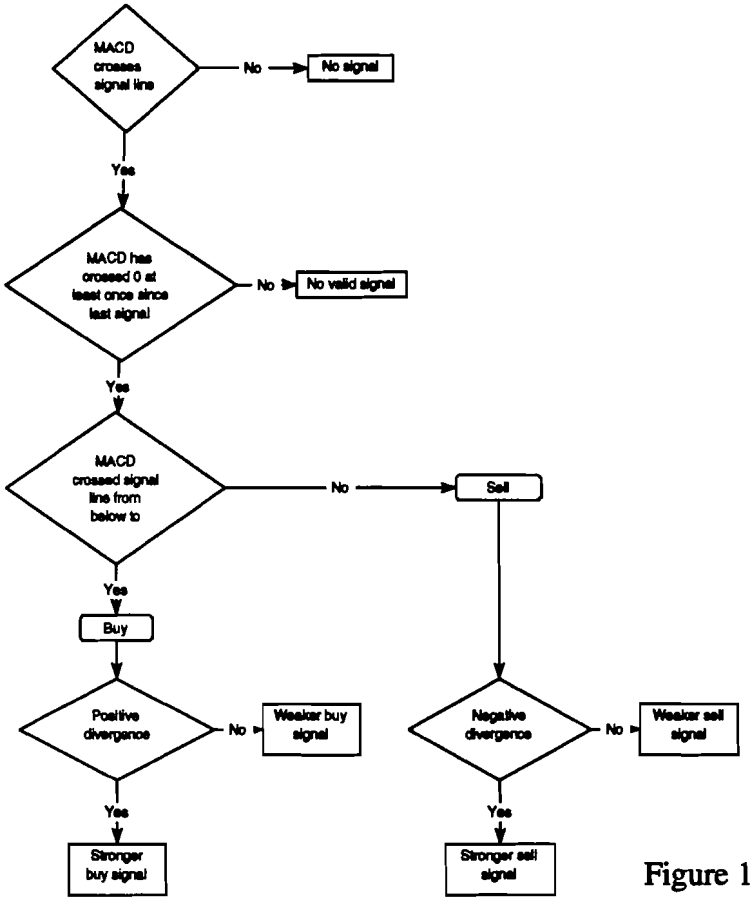


Figure 14

Positive Divergence: A positive divergence exists when prices fall to a new low, but MACD fails to make a new low along with declining price movement.
Negative Divergences exist when prices move to new highs buy MACD fails to make a new peak along with price.

• *On-Balance Volume*

OBV [AIQ 1991] is a widely used indicator that shows accumulation and distribution action. The indicator is computed as a continuous summation of daily volume. On days when prices advance, the volume for that day is added to the running total. On days when prices decline, the volume for that day is subtracted from the running total. OBV assumes that if the price today is higher than the price yesterday, all of today's volume is accumulation. If the price today is lower than the price yesterday, all of the volume is distribution.

$OBV=OBV_{t-1} + f_t V_t$

$f_t = 1.0$ if $pc_t > Pc_{t-1}$

$f_t = -1.0$ if $pc_t \leq Pc_{t-1}$

$f_t = 0.0$ if $pc_t = Pc_{t-1}$

Where:

OBV	= On-Balance Volume
f_t	= Weighting factor, day t
Pc_t	= Closing price, day t
V_t	= Volume, day t

The index above is one example of a so-called technical indicator that stock and commodity traders use in their effort to determine the market behavior. In the indicator mentioned, the basic idea is that in the long-term, there should be as many shares sold in a falling market as there are bought in a rising market. The indicator is a crude approximation to see which side is out of balance.

• *Linear Regression*

Linear regression is a standard statistical trend following method. It is defined very simply as follows:

$$y = a + bx$$

where y is the closing price and x is the time and

$$a = \frac{1}{n}(\Sigma y - b\Sigma x)$$

$$b = \frac{n\Sigma xy - \Sigma x\Sigma y}{n\Sigma x^2 - (\Sigma x)^2}$$

The semantics of this indicator are fairly trivial, nevertheless it is widely used as a first crude approximation on what to expect from the market behavior. It says that if the slope of the regression line is positive, the market tends to rise, if the slope of the line is negative, the market tends to fall. Below is the NYSE index with a regression line on the graph:

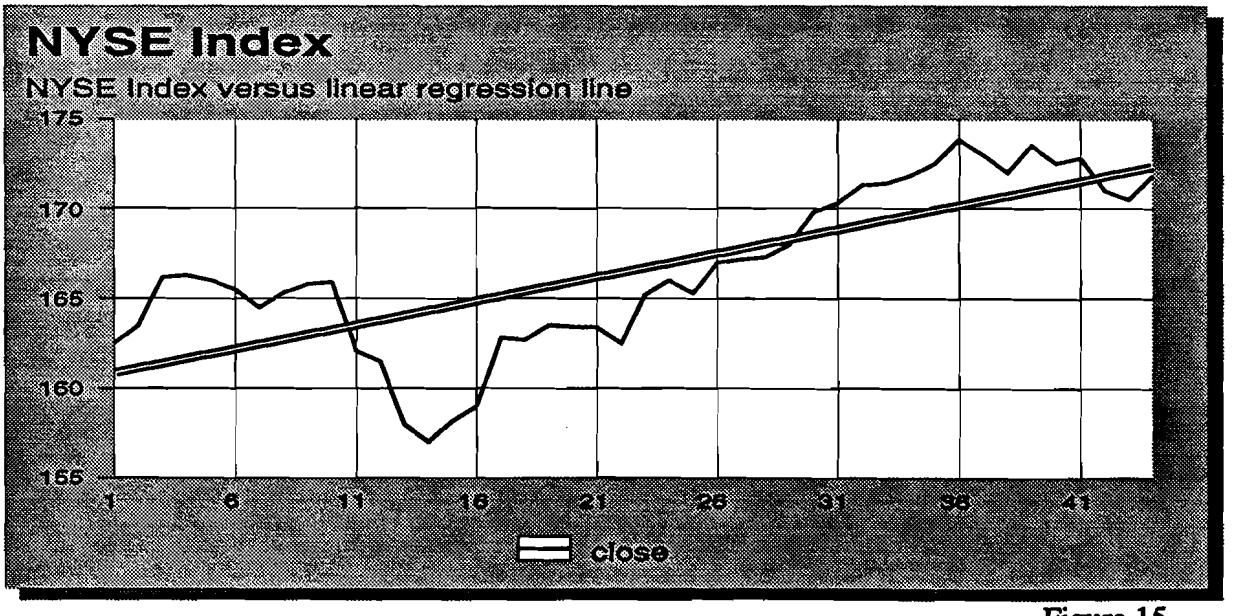


Figure 15

By plotting the endpoint of the line for any given *n*-day long regression line we would yield a curve that is related to a moving average.

• *Volume Reversal*

The Volume Reversal technique[1] is based on the concept that volume precedes price and, therefore, changes in the trend of prices often can be signaled by the expansions and contraction of volume. We first define the following terms:

- *Rally Day*: A day when the intra-day high is higher than the previous day's high and the intra-day low is the same or higher than the previous day's low.
- *Reaction Day*: A day when the intra-day low is lower than the previous day's low and the intra-day high is the same or lower than the previous day's high.
- *Inside Day*: A day when the intra-day high is the same or lower than the previous day's high and the intra-day low is the same or higher than the previous day's low.
- *Outside Day*: A day when the intra-day high is higher than the previous day's high and the intra-day low is lower than the previous day's low.

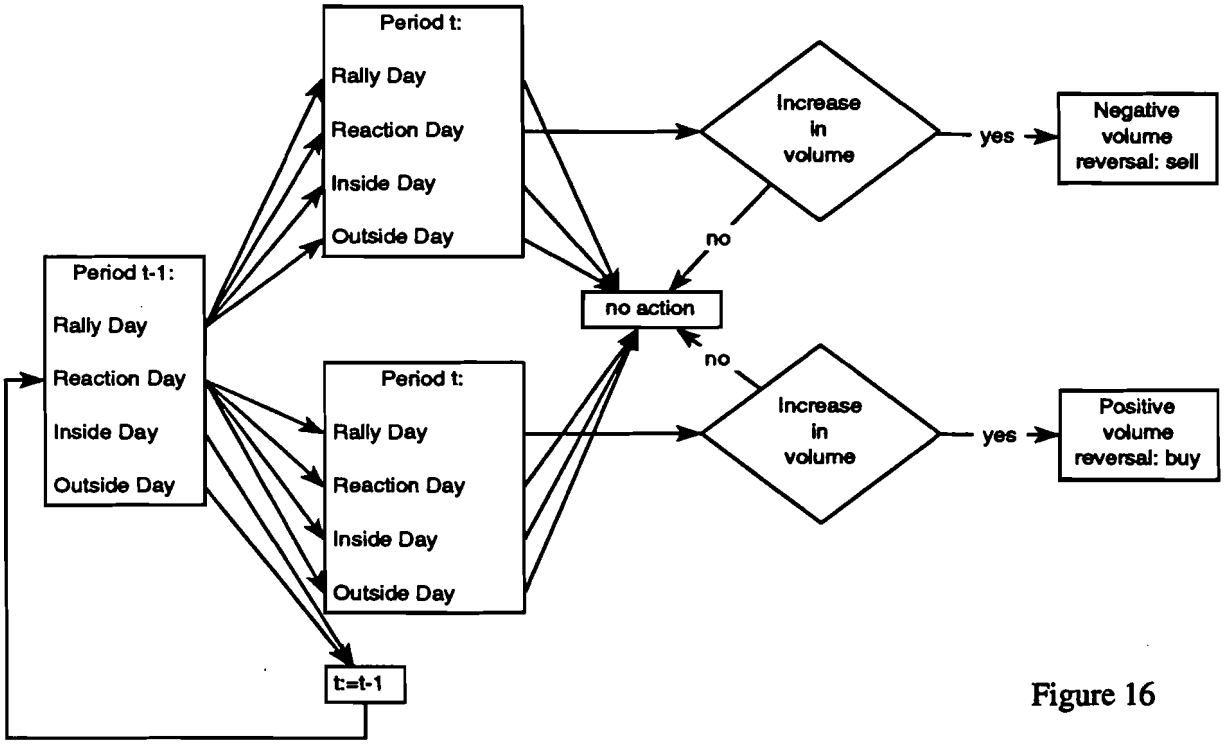


Figure 16

A Volume Reversal occurs when a change from a rally day to a reaction day, or vice versa, is accompanied by an increase in volume. Thus, if volume increases and the criteria for a reaction day are met, it is considered a negative Volume Reversal and time to sell. If, on the other hand, volume increases and the criteria for a rally day are met, it is considered a positive Volume Reversal and time to buy. Inside and outside days are ignored in the Volume Reversal technique.

1. Implemented and refined by Mark Leibovit, editor of *The Volume Reversal Survey* market newsletter.

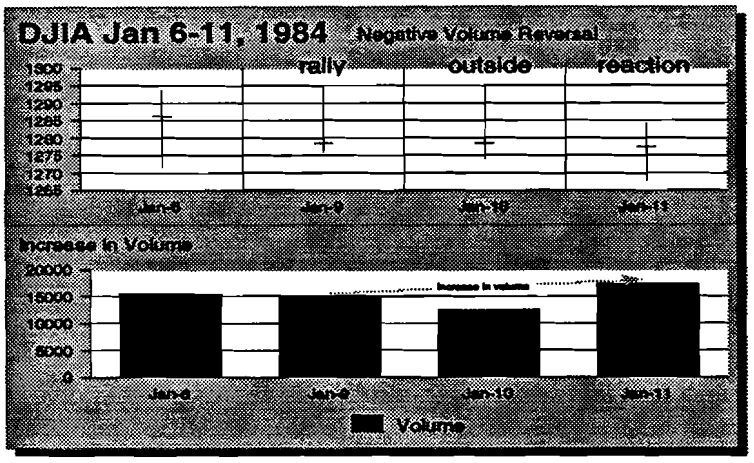


Figure 17

5.2.2.2 Rule Functions

Creating rule functions to be used by Rules to update their output, is a straightforward process. Most of the functions written in support of this project were weighting functions that combine the Inputs in a mostly linear way. In addition, we use some special-purpose mapping functions for Inputs like the weekday (Monday Rule) or the election year. Explicit negation is also supported; currently, in order to negate an Input, one needs to assign a *none*-Input as the second Input to the negate rule. The *none*-Input is a standard-normal distribution and, given independence, does not influence the output of the *negate*-Rule (or any other unary "Rule").

5.2.2.3 Joint Probability Density Functions

Throughout our sample system, we assume independence of the Inputs to the Rules. Of course, this is not realistic, especially considering the limited amount of basis information that we have at hand to derive our indicators. Nevertheless, it is the best we can do without extensive research on the interdependencies of the various indicators.

We note that finding joint probabilities and especially joint probability density functions is a hard problem in general and not particular to our approach.

5.3 Preprocessing of Indicators

To successfully handle the huge amounts of data involved in financial markets time series analysis, we take advantage of a data structure described in [Antony & Merk 1991]. Below we describe how we apply the concepts to our project.

5.3.1 Implementation of Indicators and Musical Scores

During the DeCAF[1] project, we came up with the idea to store financial data in musical scores. Although we are not aware of anyone else who takes a similar approach, we believe that both - financial data and musical scores - have many attributes in common. Their common base is that they are both strongly time-dependent. The manuals describing Objective-C classes provided by NeXT Computers, Inc., include hundreds of pages related to *"Sound, Music and Signal Processing."* We choose to take advantage of the implementation already provided and extend it to suit our needs. A simplified model of the Score object looks as follows:

1. DEsign in Computer Aided Finance

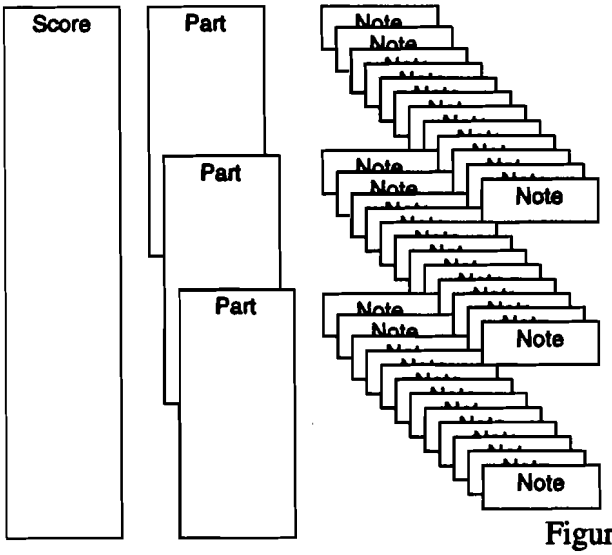


Figure 18

A Score object contains a List of Parts; a Part contains Notes. The Notes contain user-defined parameters with their respective values, if available.

Score	
Parts	The Parts represent a specific index or security. Presently, we use only one Part per score.
Notes	A Note originally contains the parameters *close, high, low, volume*

Figure 19

• *Implementing Indicators ≡ Adding Parameters*

We can attach an indicator to an index by adding information to the Notes. We can add information by adding a parameter to every single Note that is *relevant* - this means that it is up to us whether any new parameter introduced is defined for all entries, i.e. Notes, or just for those we have a signal for.

5.3.2 Caching of Values

The possibility to add parameters to Notes, is heavily used by us to cache values computed from either basic parameters or other parameters directly or indirectly computed from the basic parameters. This saves a lot of processing time, especially in those cases where we need to access parameters that are computed recursively with respect to time, or that are composed of other derived parameters. In the following, we describe how a *x-day Exponential Moving Average* is implemented:

Moving averages are important components of many indicators - the *MACD*, for example, is dependent on the difference between two moving averages of varying time spans. An exponential

moving average is used as an approximation to a true moving average, but a lot easier to compute since it is only dependent on the previous day's average and today's data. It is defined as follows:

New Exponential Average = Smoothing Constant *Today's data-Yesterday's Expo)+Yesterday's Expo
Smoothing Constant = 2/(1+x), where x is the number of days the average is applied to.

If a Note receives the request to return the value of its x-day exponential moving average for a specific parameter at its time tag, it checks whether such a value has been computed previously. If it exists, the value is returned. If it does not exist, the Note recursively calls its predecessors to receive the previous entry's information in order to compute the average. The recursion stops either when the requested parameter has been found or when the beginning of the data set is reached; in the latter case, the non-averaged value is returned.

5.3.3 Date and Time

In order to keep maximum compatibility and portability we try to stay close to the UNIX and VMS time specifications. The time is expressed in seconds since midnight (0 hour), January 1, 1970 GMT. We will record time before January 1, 1970 using negative numbers. All entries will be normalized to GMT in order to maintain relationships that are important to track concurrency especially in issues related to global arbitrage.

5.4 DeTerminator on the Test Stand

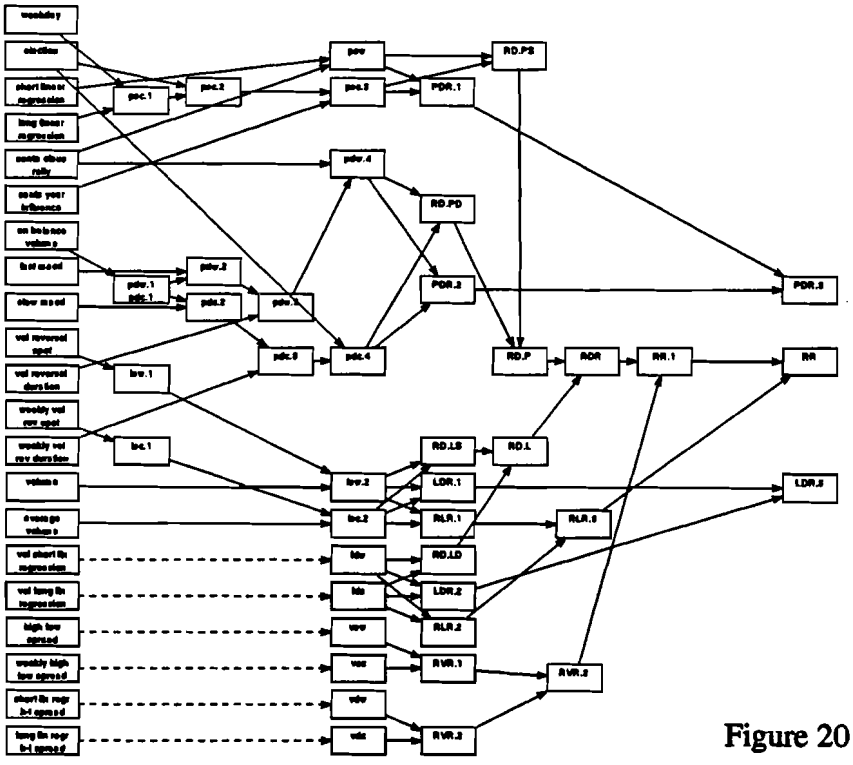


Figure 20

Above is a diagram depicting the relationships in our knowledge base to give an idea of the level of complexity. In the following, we look at how DeTerminator behaves under some specific market conditions.

5.4.1 The Crash

On October 19, 1987, the DJIA closed at 1738.74, 508 points or 22.61% lower than the previous trading day. The volume is at an all-time record of 604,330,000 million shares traded.

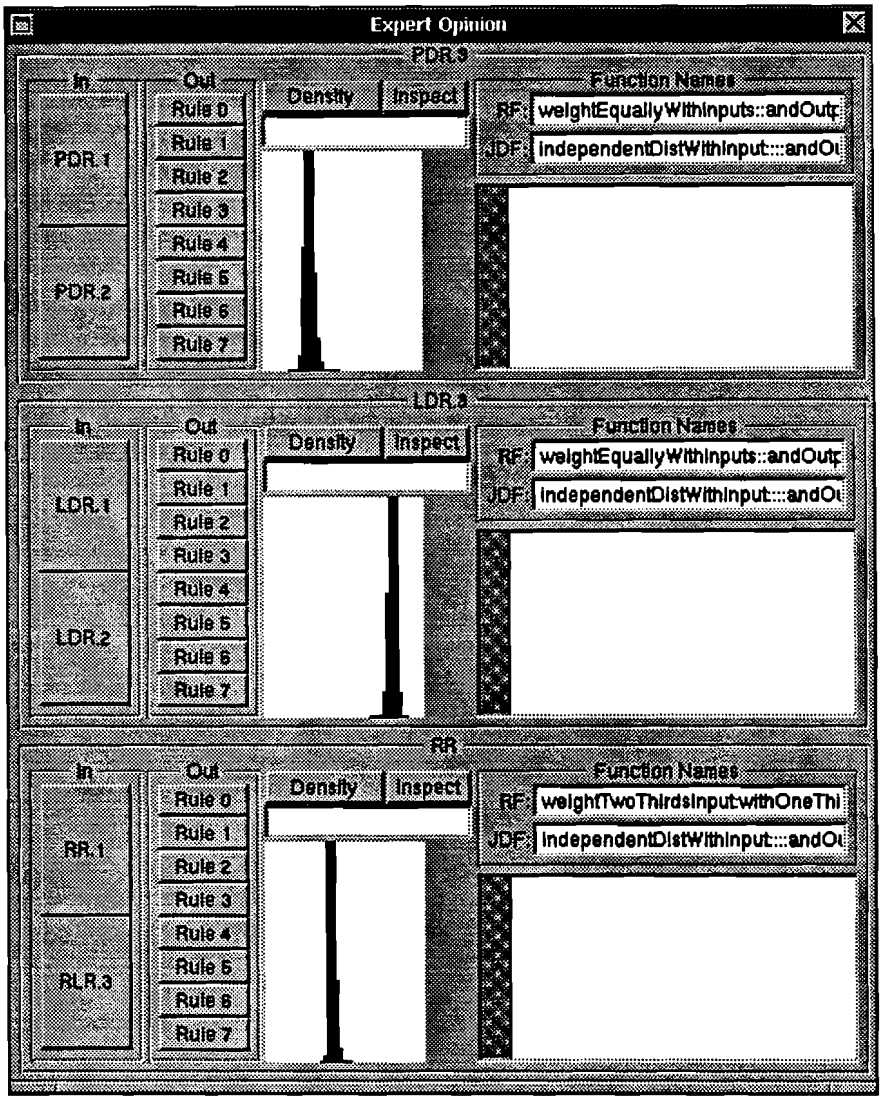


Figure 21

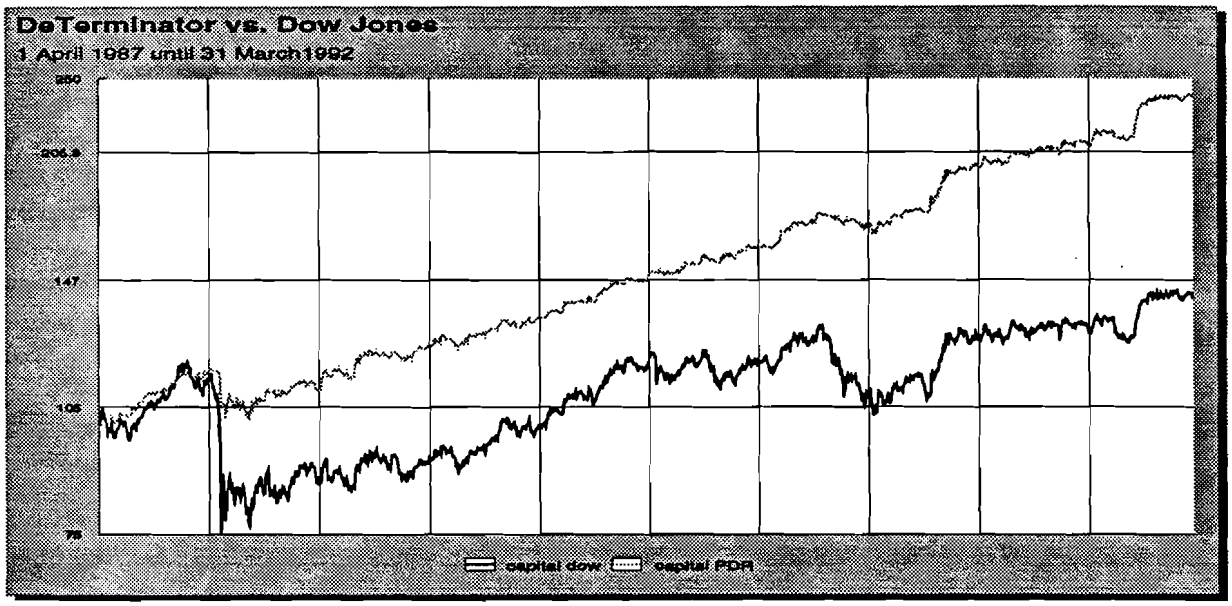
On Friday, October 17, 1987, DeTerminator predicts a LOW PDR with high probability, a HIGH LDR with high probability and a relatively low RR with high probability. In words, the recommendation is to have an investment strategy that is suitable for a market moving to a low price with high liquidity; DeTerminator sees relatively little risk in pursuing such a strategy.
See figure 21.

5.4.2 Overall Performance

Since we have not specified an exact trading strategy based upon DeTerminator's recommendations, it is difficult to express in numbers how well it performs.

To satisfy our and the reader's curiosity, however, we could not resist evaluating the performance of our Rule network. Our strategy focused on the highest peak of the price direction recommendation (PDR): have a LOW PDR-peak refer to not invested and a HIGH PDR-peak refer to

be 100% invested with corresponding, proportional investments for peaks in-between. Then, by adjusting the investment on a daily basis, we achieve a performance relative to the DJIA as shown in the graph below (figure 22). By adjusting the investment daily, we would incur fairly high commission costs, in practice. We do not, however, put the non-invested cash into a cash fund to earn interest. The graph indicates that this particular strategy combined with the Rule set is good at avoiding sharp down-turns of the market; during other periods, it seems like the strategy keeps up with the market.



The y-axis shows % of original capital at any given point in time

Figure 22

5.4.3 Dealing with Ambivalent Information, a Special Test Case

During the week of January 14, 1991, the public did not know whether the United States would start a war against Iraq with the goal to liberate Kuwait. One might have designed a Rule that captures the expectations for the case a war broke out versus if no war broke out. The *hussein*-Rule below is enabled in the days before the ultimatum runs out:

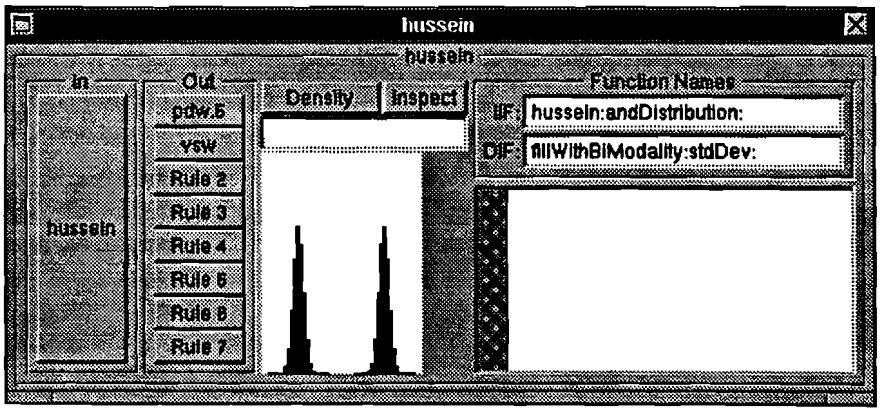


Figure 23

The Rule states that whatever is going to happen, it is either going to be very positive or very negative. The complete recommendation is shown further below.

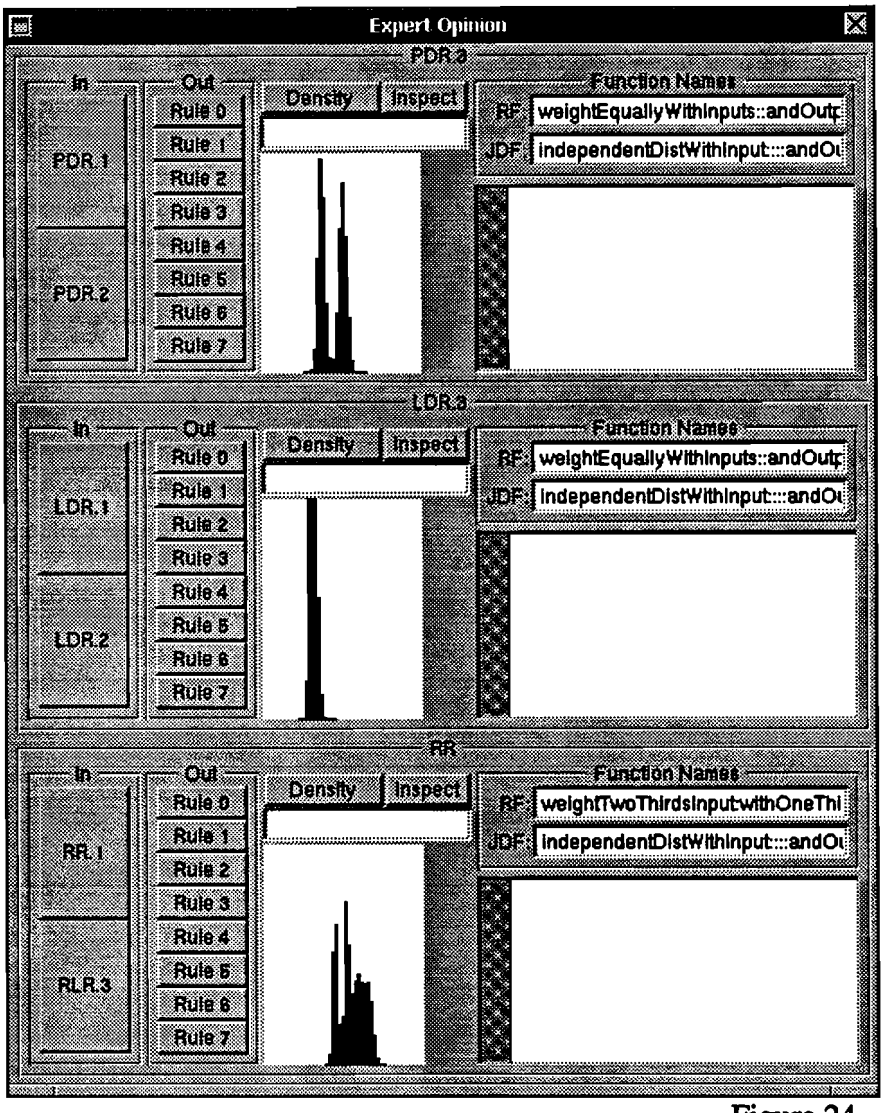


Figure 24

The network propagated the bimodality all the way to the recommendation layer of the network. According to the recommendation, a strategy accounting for the high possibility of a small market up-move as well as a high possibility for a sharp market-downturn combined with low liquidity in a risk environment that is spread from average to a little high.

In truth, the market raced up as soon as the war had broken out. We believe that the temporary addition of the *hussein*-Rule proved valuable. The remainder of the network was prevalent enough, however, to weaken the effect of the *hussein*-Rule.

6 Conclusion

We have been successful in creating a modelling tool with clearly defined semantics. It can deal with multivalent data as well as it can cope with probabilities. Our original aim to express a third dimension, namely risk, is supported under two possible scenarios: if one believes risk to be a mere matter of variance, it is directly reflected in the joint probability density functions with which our system works; if, however, one believes that risk is an orthogonal, third dimension influencing decisions under uncertainty, we can express risk as a second order property within the given framework.

Our ability to dynamically trade-off precision versus computing time as well as the choice of toll evaluation strategies enables us to use the system both in an off-line as well as in an on-line, near real-time mode.

By leveraging on the advantages of object-oriented development, we were able to not only implement a sample decision support system within a relatively short time-frame, but in addition to that, we managed to implement a tool that allows the knowledge engineer to build user-friendly decision support systems with a graphical user-interface by means of visual programming.

Due to the new concept that adds dimensionality, knowledge engineers might encounter problems as well with understanding on how to use these additional possibilities to create more powerful decision support systems, as with the dimensionality reduction and final representation of the output of a decision support system based upon our tool-kit.

Nevertheless, the fact that we could implement a relatively primitive decision support system with our tool-kit in a very short time-frame and that without refining the first, relatively arbitrary, knowledge base, we already could achieve performance in excess the market's.

Overall, we are pleased to have created a foundation for a computational treatment of distribution based knowledge.

7 Outlook

Although we are pleased with our achievements, there is, of course, plenty of room for further development and improvement. A couple of ideas that should be pursued further, shall be briefly discussed here:

It is worthwhile investigating whether a learning algorithm could be employed to either learn a knowledge base from scratch or to optimize an existing knowledge base created by a knowledge engineer. We believe that the most promising approach probably is based on some sort of gradient descent and error back-propagation method, which implies that our rule functions as well as our joint probability density functions be continuous and describable by means a limited set of parameters.

This would also have another advantage - the few parameters of such joint probability density functions and rule functions could be controlled by direct user interaction and therefore alleviate the need to define rule functions and joint probability density functions in terms of c-code.

An improvement in the results of the numerical algorithms could be achieved by using a dynamically adapting sample interval size that depends on the slope of the probability density function at the location of that interval; thus, steeper areas would be assigned more bins that flatter areas. In an ideal case, one could ensure that the absolute difference in density from one bin to the next is always the same. This would simplify the computation joint probability density functions.

Some less significant, but still desirable addition would be an algorithm that creates a visual representation of our knowledge base automatically by tracing all the connections among the instantiated building blocks.

Finally, we should apply DeTerminator to a complex task that takes extensive advantage of multivalent information.

Also, we look forward to the availability of NeXTstep 3.0 which should allow us to create an even more intuitive user-interface for our tool-kit since what we are doing has no longer to be achieved by means of some sneaky tricks, but rather is fully supported and *documented*, in the next version of InterfaceBuilder.

8 Acknowledgments

First and foremost, our thanks go to Leslie P. Kaelbling, who proved to be a highly-motivating and highly-motivated advisor always available for constructive criticism and moral support. In addition, she proved to be a humorous person who is great fun to work with.

We are thankful to John Irwin of Franz, Inc., for spending days of his time to narrow down an operating system bug. Also, Julie Zelenski from NeXT Computer, Inc., was very helpful in showing us how to work around some undocumented features of InterfaceBuilder and for being generally one of the most helpful people at NeXT developer's support.

Furthermore, we would like to thank all those that have helped make DeCAF a success in previous years; these include Jak Kirman, Jerome Stein, F. Canova, John Savage, P. Falb and F. Bisshopp.

We want to thank Kim and Hanna for not showing their jealousy on machnix and idefix that were happy spending long and late hours in our company and whose heart beat for us reached 25MHz. Also, we want to thank our parents for their ongoing support, even if at times, they must have felt like birds feeding their children that keep starring with a wide open beak.

Last, but not least, we thank Tom Dean for suggesting to us to do something completely orthogonal to what we have done so far.

9 Appendices

9.1 Bibliography

[AIQ 1991] *AIQ IndexExpert*, AIQ Systems, Inc., Technical Appendix IX-16

[AI/WS 1991] *Proceedings of the First International Conference on Artificial Intelligence Applications on Wall Street*, IEEE Computer Society Press 1991

[Antony & Merk 1991] *DEsign in Computer Aided Finance, DECAF*, by Ronald Antony and Axel Merk, Brown University, 1991.

[Babylon 1989] *The AI Workbench Babylon*, Addison-Wesley, 1989 (English draft-version)

[Colby 1988] *The Encyclopedia Of Technical Market Indicators*, Robert Colby and Thomas Meyers, Dow Jones-Irwin, 1988

[Fama 1976] chapter 5, *Foundations of Finance*, Eugene F. Fama, Basic Books, Inc., 1976

[Hirsch 1986] *Don't Sell Stocks On Monday*, Yale Hirsch, Penguin Books, 1986

[Mandelbrot 1964] "The Variation of Certain Speculative Prices", in P. Cootner, ed., *The Random Character of Stock Prices*. Cambridge, MA: M.I.T. Press, 1964

[Nordhaus 1989] *Economics*, Paul Samuelson and William Nordhaus, McGraw Hill 1989, 13th edition

[NeXT 1992] *DB-KIT: Developing Object-Oriented Database Applications, Eighth In The NeXT Computer, Inc., White Paper Library.* NeXT Computer, Inc., 1992.

[Peters 1991] *Chaos and Order in the Capital Markets*, Edgar E. Peters. John Wiley & Sons, Inc., 1991

```
#import <appkit/ButtonCell.h>

@interface ButtonCell(Actions)

- propagateChange:sender;
- batchMode:sender;

@end
```

```
#import "Actions.h"

@implementation ButtonCell(Actions)

- propagateChange:sender
{
    if([self target]==nil)
      return self;
    else
      return [[self target] propagateChange:sender];
}

- batchMode:sender
{
    if([self target]==nil)
      return self;
    else
      return [[self target] batchMode:sender];
}

@end
```

```
#import <nib/InterfaceBuilder.h>

@interface BinButtonCellInspector:Inspector
{
    id        foregroundGraySlider;
    id        foregroundGrayTextField;
    id        backgroundGraySlider;
    id        backgroundGrayTextField;
    id        maxGraphValueSlider;
    id        maxGraphValueTextField;
    id        binValueSlider;
    id        binValueTextField;
    id        disabledSwitch;
    id        tagTextField;
}

+ finishLoading:(struct mach_header *)header;
+ startUnloading;
- init;
- ok:sender;
- revert:sender;

@end
```

```
#import "BinButtonCellInspector.h"
#import "BinButtonCell.h"
#import <appkit/Application.h>
#import <appkit/Slider.h>
#import <appkit/TextField.h>

@implementation BinButtonCellInspector

+ finishLoading:(struct mach_header *)header
{
    NIBDidLoadClass(self, header);
    return nil;
}

+ startUnloading
{
    NIBWillUnloadClass(self);
    return nil;
}

- init
{
    [super init];
    [NXApp loadNibSection:"BinButtonCellInspector.nib" owner:self];
    return self;
}

- ok:sender
{
    [object setForegroundGray:[foregroundGraySlider floatValue]];
    [object setBackgroundGray:[backgroundGraySlider floatValue]];
    [object setMaxGraphValue:[maxGraphValueSlider floatValue]];
    [object setBinValue:[binValueSlider doubleValue]];
    [object setEnabled:(![disabledSwitch state])];
    [object setTag:[tagTextField intValue]];
    return [super ok:sender];
}

- revert:sender
{
    [foregroundGraySlider setFloatValue:[object foregroundGray]];
    [foregroundGrayTextField setFloatValue:[object foregroundGray]];
    [backgroundGraySlider setFloatValue:[object backgroundGray]];
    [backgroundGrayTextField setFloatValue:[object backgroundGray]];
    [maxGraphValueSlider setFloatValue:[object maxGraphValue]];
    [maxGraphValueTextField setFloatValue:[object maxGraphValue]];
    [binValueSlider setDoubleValue:[object binValue]];
    [binValueTextField setDoubleValue:[object binValue]];
    [disabledSwitch setState:(![object isEnabled])];
    [tagTextField setIntValue:[object tag]];
    return [super revert:sender];
}

@end
```

```
#import <appkit/graphics.h>
#import <dpsclient/wraps.h>
#import <appkit/ButtonCell.h>

//todo:
//will be changed to use NXColor
//add methods to use color in addition to grayValues

@interface BinButtonCell:ButtonCell
{
    float backgroundGrayValue;
    float foregroundGrayValue;
    float maxGraphValue;
    double binValue;
}

- (id)init;
- (id)setMaxGraphValue:(float)aGraphValue;
- (float)maxGraphValue;
- (id)setBackgroundGray:(float)aGrayValue;
- (float)backgroundGray;
- (id)setForegroundGray:(float)aGrayValue;
- (float)foregroundGray;
- (id)setBinValue:(double)aBinValue;
- (double)binValue;
- (id)drawInside:(const NXRect *)aRect inView:(id)controlView;
- takeForegroundGrayFrom:sender;
- takeBackgroundGrayFrom:sender;
- takeMaxGraphValueFrom:sender;
- takeBinValueFrom:sender;
- (const char*)inspectorName;
- write:(NXTypedStream *)stream;
- read:(NXTypedStream *)stream;

@end
```

```
#import "BinButtonCell.h"
#import <appkit/View.h>
#import <appkit/Control.h>
#import "DensityButtonCell.h"

@implementation BinButtonCell

- (id)init
{
    [super init];
    backgroundGrayValue=1.0; //NX_WHITE;
    foregroundGrayValue=0.0; //NX_BLACK;
    binValue=(double)0.5;
    maxGraphValue=1.0;
    return self;
}

- (id)setMaxGraphValue:(float)aGraphValue
{
    maxGraphValue=aGraphValue;
    return self;
}

- (float)maxGraphValue
{
    return maxGraphValue;
}

- (id)setBackgroundGray:(float)aGrayValue
{
    backgroundGrayValue=aGrayValue;
    return self;
}

- (float)backgroundGray
{
    return backgroundGrayValue;
}

- (id)setForegroundGray:(float)aGrayValue
{
    foregroundGrayValue=aGrayValue;
    return self;
}

- (float)foregroundGray
{
    return foregroundGrayValue;
}

- (id)setBinValue:(double)aBinValue
{
    binValue=aBinValue;
    return self;
}

- (double)binValue
{
    return binValue;
}

- (id)drawInside:(const NXRect *)aRect inView:(id)controlView
{
    NXRect barRe     *aRect;
    float factor;

    [controlView lockFocus];
    PSsetgray([self backgroundGray]);
    NXRectFill(aRect);
    PSsetgray([self foregroundGray]);
    //modify the size of barRect here...
    factor = [self binValue]/maxGraphValue;
    if((factor<=1.0)&&(factor>=0.0))
        barRect.size.height*=factor;
    //this is to accomodate for the flipped coordinate system..
    //maybe there is some other way to do that with setFlipped:...
    barRect.origin.y=(*aRect).origin.y+(*aRect).size.height-barRect.size.height;
    NXRectFill(&barRect);
    [controlView unlockFocus];
    return self;
}

- takeForegroundGrayFrom:sender;
{
    if([sender respondsTo:@selector(grayValue)])
        [self setForegroundGray:(float)[sender grayValue]];
    else
        [self setForegroundGray:[sender floatValue]];
    [[self controlView] updateCell:self];
    return self;
}

- takeBackgroundGrayFrom:sender;
{
    if([sender respondsTo:@selector(grayValue)])
        [self setBackgroundGray:(float)[sender grayValue]];
    else
        [self setBackgroundGray:[sender floatValue]];
    [[self controlView] updateCell:self];
    return self;
}

- takeMaxGraphValueFrom:sender;
{
    if([sender respondsTo:@selector(maxGraphValue)])
        [self setMaxGraphValue:[sender maxGraphValue]];
    else
        [self setMaxGraphValue:[sender floatValue]];
    [[self controlView] updateCell:self];
    return self;
}

- takeBinValueFrom:sender;
{
    if([sender respondsTo:@selector(binValue)])
        [self setBinValue:[sender binValue]];
    else
        [self setBinValue:[sender doubleValue]];
    [[self controlView] updateCell:self];
    return self;
}

- (const char*)inspectorName
{
    return "BinButtonCellInspector";
}

rite:(NXTypedStream *)stream
```

```
{
    [super write:stream];
    NXWriteTypes(stream,
                 "fffd",
                 &backgroundGrayValue,
                 &foregroundGrayValue,
                 &maxGraphValue,
                 &binValue);
    return self;
}

- read:(NXTypedStream *)stream
{
    [super read:stream];
    NXReadTypes(stream,
                "fffd",
                &backgroundGrayValue,
                &foregroundGrayValue,
                &maxGraphValue,
                &binValue);
    return self;
}

@end
```

```
#import <appkit/Button.h>

//todo:
//will be changed to use NXColor
//add methods to use color in addition to grayValues

@interface BinButton:Button
{
}

//+ initialize; //see history in BinButton.m
- init;
- initFrame:(const NXRect *)frameRect;
- initFrame:(const NXRect*)frameRect
  title:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag;
- initFrame:(const NXRect *)frameRect
  icon:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag;
- takeForegroundGrayFrom:sender;
- takeBackgroundGrayFrom:sender;
- takeMaxGraphValueFrom:sender;
- takeBinValueFrom:sender;

@end
```

```
#import "BinButton.h"
#import "BinButtonCell.h"
#import "DensityButtonCell.h"
#import <appkit/View.h>

@implementation BinButton

// Here some interesting history:
// First what we wanted to do:
//+ initialize
//{
//        [super initialize];
//        [self setCellClass:[BinButtonCell class]];
//        return self;
//}
//
// Now what it should be, since super is initialized automatically
// by the runtime system, if need be:
//+ initialize
//{
//    // to prevent interference with calls from subclasses:
//    if ( self == [BinButton class] )
//    {
//        [self setCellClass:[BinButtonCell class]];
//    }
//    return self;
//}
//
// Now, we realize that cellClass is not really a class variable, but a
// static variable, globally bound :-(
// So we have to set it temporarily during the init methods of the
// instances...
// Here is what we would like to do:
//- init
//{
//  id someCellClass;
//
//  someCellClass=[[BinButton class] setCellClass:[BinButtonCell class]];
//  [super init];
//  [[BinButton class] setCellClass:someCellClass];
//  return self;
//}
//
// Now the last sad discovery: +setCellClass does not return the previous
// cellClass, thus we have to hard code as follows:

- init
{
    [[BinButton class] setCellClass:[BinButtonCell class]];
    [super init];
    [[BinButton class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
{
    [[BinButton class] setCellClass:[BinButtonCell class]];
    [super initFrame:frameRect];
    [[BinButton class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
  title:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag
{
    [[BinButton class] setCellClass:[BinButtonCell class]];
    [super initFrame:frameRect
           title:aString
           tag:anInt
           target:anObject
           action:aSelector
           key:charCode
           enabled: flag];
    [[BinButton class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
  icon:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag
{
    [[BinButton class] setCellClass:[BinButtonCell class]];
    [super initFrame:frameRect
           icon:aString
           tag:anInt
           target:anObject
           action:aSelector
           key:charCode
           enabled: flag];
    [[BinButton class] setCellClass:[ButtonCell class]];
    return self;
}

- takeForegroundGrayFrom:sender;
{
//    if([sender respondsTo:@selector(grayValue)])
//      [cell setForegroundGray:(float)[sender grayValue]];
//    else
//      [cell setForegroundGray:[sender floatValue]];
    [[self selectedCell] takeForegroundGrayFrom:sender];
    return [self display];
}

- takeBackgroundGrayFrom:sender;
{
//    if([sender respondsTo:@selector(grayValue)])
//      [cell setBackgroundGray:(float)[sender grayValue]];
//    else
//      [cell setBackgroundGray:[sender floatValue]];
    [[self selectedCell] takeBackgroundGrayFrom:sender];
    return [self display];
}

- takeMaxGraphValueFrom:sender;
{
//    if([sender respondsTo:@selector(maxGraphValue)])
//      [cell setMaxGraphValue:[sender maxGraphValue]];
//    else
//      [cell setMaxGraphValue:[sender floatValue]];
```

```
    [[self selectedCell] takeMaxGraphValueFrom:sender];
    return [self display];
}

- takeBinValueFrom:sender;
{
//    if([sender respondsTo:@selector(binValue)])
//      [cell setBinValue:[sender binValue]];
//    else
//      [cell setBinValue:[sender doubleValue]];
    [[self selectedCell] takeBinValueFrom:sender];
    return [self display];
}

@end
```

```
#import <appkit/Matrix.h>

//todo:
//will be changed to use NXColor
//add methods to use color in addition to grayValues

@interface BinMatrix:Matrix
{
}

//+ initialize; //see history in BinButton.m
- init;
- initFrame:(const NXRect *)frameRect;
- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  prototype:aCell
  numRows:(int)rowsHigh
  numCols:(int)colsWide;
- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  cellClass:factoryId
  numRows:(int)rowsHigh
  numCols:(int)colsWide;
- takeForegroundGrayFrom:sender;
- takeBackgroundGrayFrom:sender;
- takeMaxGraphValueFrom:sender;
- takeBinValueFrom:sender;

@end
```

```
#import "BinMatrix.h"
#import "BinButtonCell.h"
#import "DensityButtonCell.h"
#import <appkit/View.h>

@implementation BinMatrix

- init
{
#ifdef PALETTE
    id someCellPrototype;
#endif

    [[BinMatrix class] setCellClass:[BinButtonCell class]];
    [super init];
    [[BinMatrix class] setCellClass:[ButtonCell class]];
#ifdef PALETTE
    someCellPrototype=[self setPrototype:[[BinButtonCell allocFromZone:[self zone]]init]
];
    if(someCellPrototype!=nil)
        {
            //[[self prototype]setSize:[someCellPrototype size]];
            [someCellPrototype free];
        }
#endif
    return self;
}

- initFrame:(const NXRect *)frameRect
{
#ifdef PALETTE
    id someCellPrototype;
#endif

    [[BinMatrix class] setCellClass:[BinButtonCell class]];
    [super initFrame:frameRect];
    [[BinMatrix class] setCellClass:[ButtonCell class]];
#ifdef PALETTE
    someCellPrototype=[self setPrototype:[[BinButtonCell allocFromZone:[self zone]]init]
];
    if(someCellPrototype!=nil)
        {
            //[[self prototype]setSize:[someCellPrototype size]];
            [someCellPrototype free];
        }
#endif
    return self;
}

- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  prototype:aCell
  numRows:(int)rowsHigh
  numCols:(int)colsWide
{
    [[BinMatrix class] setCellClass:[BinButtonCell class]];
    [super initFrame:frameRect
           mode:aMode
           prototype:aCell
           numRows:rowsHigh
           numCols:colsWide];
    [[BinMatrix class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  cellClass:factoryId
  numRows:(int)rowsHigh
  numCols:(int)colsWide
{
    [[BinMatrix class] setCellClass:[BinButtonCell class]];
    [super initFrame:frameRect
           mode:aMode
           cellClass:factoryId
           numRows:rowsHigh
           numCols:colsWide];
    [[BinMatrix class] setCellClass:[ButtonCell class]];
    return self;
}

- takeForegroundGrayFrom:sender;
{
//    if([sender respondsTo:@selector(grayValue)])
//      [[self selectedCell] setForegroundGray:(float)[sender grayValue]];
//    else
//      [[self selectedCell] setForegroundGray:[sender floatValue]];
    [[self selectedCell] takeForegroundGrayFrom:sender];
    return [self display];
}

- takeBackgroundGrayFrom:sender;
{
//    if([sender respondsTo:@selector(grayValue)])
//      [[self selectedCell] setBackgroundGray:(float)[sender grayValue]];
//    else
//      [[self selectedCell] setBackgroundGray:[sender floatValue]];
    [[self selectedCell] takeBackgroundGrayFrom:sender];
    return [self display];
}

- takeMaxGraphValueFrom:sender;
{
//    if([sender respondsTo:@selector(maxGraphValue)])
//      [[self selectedCell] setMaxGraphValue:[sender maxGraphValue]];
//    else
//      [[self selectedCell] setMaxGraphValue:[sender floatValue]];
    [[self selectedCell] takeMaxGraphValueFrom:sender];
    return [self display];
}

- takeBinValueFrom:sender;
{
//    if([sender respondsTo:@selector(binValue)])
//      [[self selectedCell] setBinValue:[sender binValue]];
//    else
//      [[self selectedCell] setBinValue:[sender doubleValue]];
    [[self selectedCell] takeBinValueFrom:sender];
    return [self display];
}

@end
```

```
/*
 *     Generated by the NeXT Interface Builder.
 */

#import <stdlib.h>
#import <nib/InterfaceBuilder.h>
#import <appkit/Application.h>

void main(int argc, char *argv[]) {
    NIBInit("CAFIB");
    NIBLoadPalette("ButtonPalette.nib", "ButtonPalette", "ButtonPaletteH");
    NIBLoadPalette("CAFPalette.nib", "CAFPalette", "CAFPaletteH");
    NIBLoadPalette("zzz.nib", "zzz", "zzzH");
    NIBRun();
}
```

```
#import <musickit/musickit.h>
#import <objc/Object.h>
#import <musickit/Note.h>
#import <float.h>
#import "CAFPart.h"
#import "globals.h"
#import "dates.h"
#import "preprocessing.h"

@interface Note (CAFNote)
// no new instance variables in category

// additional methods
- (double)infoAsDouble:(int)par;
- (BOOL)isInfoPresent:(int)par;
- (double)maxWithinDays:(unsigned int)days forParName:(char *)aName;
- (double)minWithinDays:(unsigned int)days forParName:(char *)aName;
- (double)expMovingAverageForDays:(unsigned int)numOfDays andParName:(char *)aName;
- (double)onBalanceVolumeForParName:(char *)aName;
- (double)macdWithShortMA:(unsigned int)shortMADays
                  andLongMA:(unsigned int)longMADays
                  andParName:(char *)aName;
- (double)macdNegOf:(double)compMACD
           forShortMA:(unsigned int)shortMADays
           andLongMA:(unsigned int)longMADays
           andParName:(char *)aName;
- (double)macdSignalForDays:(unsigned int)signalDays
                  andShortMA:(unsigned int)shortMADays
                  andLongMA:(unsigned int)longMADays
                  andParName:(char *)aName;
- (double)macdDiffForDays:(unsigned int)signalDays
                  andShortMA:(unsigned int)shortMADays
                  andLongMA:(unsigned int)longMADays
                  andParName:(char *)aName;
- (double)macdDecisionForDays:(unsigned int)signalDays
                  andShortMA:(unsigned int)shortMADays
                  andLongMA:(unsigned int)longMADays
                  andParName:(char *)aName;
- (double)macdDecisionForDays:(unsigned int)signalDays
                  andShortMA:(unsigned int)shortMADays
                  andLongMA:(unsigned int)longMADays
                  andParName:(char *)aName
                  depreciated:(double)percentage;
- (double)sumForParName:(char *)aName;
- (double)sumOfProductForParName:(char *)aName andParName:(char *)aName;
- (double)sumOfSquaresForParName:(char *)aName;
- (double)sumForDays:(unsigned int)days
                  andParName:(char *)aName;
- (double)sumOfProductForDays:(unsigned int)days
                  andParName:(char *)aName
                  andParName:(char *)otherName;
- (double)sumOfSquaresForDays:(unsigned int)days
                  andParName:(char *)aName;
- (double)linearRegressionCoefficientForDays:(unsigned int)days
                  andXPar:(char *)xPar andYPar:(char *)yPar;
- (double)linearRegressionOffsetForDays:(unsigned int)days
                  andXPar:(char *)xPar andYPar:(char *)yPar;
- (int)marketType;
- (int)marketTypeDays:(unsigned int)days;
- (int)increaseRelativeToNote:aNote forDays:(unsigned int)days forParName:(char *)aName;
- (int)increaseRelativeToNote:aNote forParName:(char *)aName;
- (int)volumeReversal;
- (int)volumeReversalDays:(unsigned int)days;
- (double)volumeReversalDays:(unsigned int)days depreciated:(double)percentage;
- (int)volumeReversalWeekly;
- (int)weekday;
- (int)year;
- (int)dayOfYear;
- (int)santaClaus;
- (int)santaClausYearInfluenceForParName:(char *)aName;
- ensureHighLowSpreadAvailableUpToNoteForDays:(unsigned int)days;
- (double)highLowSpread:(unsigned int)days;
@end
```

```
#import "CAFNote.h"

@implementation Note (CAFNote)

- (double)infoAsDouble:(int)par
{
  char *parameterName=[Note nameOfPar:par];

  if(strcmp(parameterName,"timeTag")==0)
      return [self timeTag];
  else
      if(strcmp(parameterName,"index")==0)
          return [[self part] indexOfNote:self];
      else
          return [self parAsDouble:par];
}

- (BOOL)isInfoPresent:(int)par
{
  if((strcmp([Note nameOfPar:par],"timeTag")==0) || (strcmp([Note nameOfPar:par],"index"
)==0))
    return YES;
  else
    return [self isParPresent:par];
}


- (double)maxWithinDays:(unsigned int)days forParName:(char *)aName
{
  int par=[Note parName:aName];
  double maximum=[self infoAsDouble:par];

  if(days>1)
    {
      id previousNote=[[self part] previous:self];

      if(previousNote!=nil)
        maximum=MAX(maximum, [previousNote maxWithinDays:(days-1) forParName:aName] );
    }
  return maximum;
}

- (double)minWithinDays:(unsigned int)days forParName:(char *)aName
{
  int par=[Note parName:aName];
  double minimum=[self infoAsDouble:par];

  if(days>1)
    {
      id previousNote=[[self part] previous:self];

      if(previousNote!=nil)
        minimum=MIN(minimum, [previousNote minWithinDays:(days-1) forParName:aName] );
    }
  return minimum;
}

- (double)expMovingAverageForDays:(unsigned int)numOfDays andParName:(char *)aName
{
  char *parName=(char *)malloc(80);
  int emaPar;
  int parameter=[Note parName:aName];

  sprintf(parName,"ema_%u_%s",numOfDays,aName);
  emaPar=[Note parName:parName];
  free(parName);

  if(numOfDays==1) /* avoid recursive search for simple case - for efficiency only */
    return [self infoAsDouble:parameter];
  else
  if([self isInfoPresent:emaPar])
    {
      return [self infoAsDouble:emaPar];
    }
   else
    {
      // recursively find appropriate value
      double newExpMovAverage;
      id previousNote=[[self part] previous:self];

      if(previousNote==nil)
        newExpMovAverage=[self infoAsDouble:parameter];
      else
        {
          double smooth=2.0/((double)(numOfDays+1));
          double prevMovingAverage=[previousNote expMovingAverageForDays:numOfDays andPar
Name:aName];

          newExpMovAverage=smooth*([self infoAsDouble:parameter] - prevMovingAverage) + p
revMovingAverage;
        }
      // store appropriate value
      [self setPar:emaPar toDouble:newExpMovAverage];
      return newExpMovAverage;
    }
}

- (double)onBalanceVolumeForParName:(char *)aName
{
  double newOBV=0.0;
  char *parName=(char *)malloc(80);
  int obvPar;

  sprintf(parName,"obv_%s",aName);
  obvPar=[Note parName:parName];
  free(parName);

  if([self isInfoPresent:obvPar])
    {
      return [self infoAsDouble:obvPar];
    }
   else
    {
      // recursively find appropriate value
      int parameter=[Note parName:aName];
      int volumePar=[Note parName:"volume"];
      id previousNote=[[self part] previous:self];

      if(previousNote!=nil)
        {
          double prevOBV=[previousNote onBalanceVolumeForParName:aName];
          double prevPar=[previousNote infoAsDouble:parameter];
          double thisPar=[self infoAsDouble:parameter];
          double volume=[self infoAsDouble:volumePar];

          prevPar*=1.001;
          newOBV=prevOBV;
          if(thisPar>prevPar)
```

```
            newOBV+=volume;
          else
            if(thisPar<prevPar)
              newOBV-=volume;
        }
      // store appropriate value
      [self setPar:obvPar toDouble:newOBV];
      return newOBV;
    }
}


- (double)macdWithShortMA:(unsigned int)shortMADays
                andLongMA:(unsigned int)longMADays
                andParName:(char *)aName
{
 return [self expMovingAverageForDays:shortMADays andParName:aName] -
        [self expMovingAverageForDays:longMADays andParName:aName];
}

/* returns timeTag of last zero-crossing */
- (double)macdNegOf:(double)compMACD
                forShortMA:(unsigned int)shortMADays
                andLongMA:(unsigned int)longMADays
                andParName:(char *)aName
{
  double thisMACD = [self macdWithShortMA:shortMADays andLongMA:longMADays andParName:aN
ame];

  if(thisMACD * compMACD < 0.0)
    return [self timeTag];
  else
    {
      id previousNote=[[self part] previous:self];

      if(previousNote==nil)
        return -DBL_MAX;
      else
        {
          return [previousNote macdNegOf:compMACD
                               forShortMA:shortMADays
                               andLongMA:longMADays
                               andParName:aName];
        }
    }
}

- (double)macdSignalForDays:(unsigned int)signalDays
                      andShortMA:(unsigned int)shortMADays
                      andLongMA:(unsigned int)longMADays
                      andParName:(char *)aName;
{
  char *parNameMacdSignal=(char *)malloc(80);
  int macdSignalPar;

  sprintf(parNameMacdSignal,"macdSignal_%u_%u_%u_%s",signalDays,shortMADays,longMADays,a
Name);
  macdSignalPar=[Note parName:parNameMacdSignal];
  free(parNameMacdSignal);

  if([self isInfoPresent:macdSignalPar])
    {
      return [self infoAsDouble:macdSignalPar];
    }
  else
    {
      // recursively find appropriate value
      double newSignal;
      id previousNote=[[self part] previous:self];

      if(previousNote==nil)
        {
        newSignal=[self macdWithShortMA:shortMADays
                     andLongMA:longMADays
                     andParName:aName];
        }
      else
        {
          double smooth=2.0/(signalDays+1);
          double prevSignal=[previousNote macdSignalForDays:signalDays
                        andShortMA:shortMADays
                        andLongMA:longMADays
                        andParName:aName];

          newSignal=smooth*([self macdWithShortMA:shortMADays
                               andLongMA:longMADays
                               andParName:aName]
                        - prevSignal) + prevSignal;
        }
      // store appropriate value
      [self setPar:macdSignalPar toDouble:newSignal];
      return newSignal;
    }
}

- (double)macdDiffForDays:(unsigned int)signalDays
                      andShortMA:(unsigned int)shortMADays
                      andLongMA:(unsigned int)longMADays
                      andParName:(char *)aName
{
  return [self macdWithShortMA:shortMADays
                andLongMA:longMADays
                andParName:aName]
         - [self macdSignalForDays:signalDays
              andShortMA:shortMADays
              andLongMA:longMADays
              andParName:aName];
}

- (double)macdDecisionForDays:(unsigned int)signalDays
                      andShortMA:(unsigned int)shortMADays
                      andLongMA:(unsigned int)longMADays
                      andParName:(char *)aName

{
  char *parName=(char *)malloc(80);
  int macdDecPar;

  sprintf(parName,"macdDec_%u_%u_%u_%s",signalDays,shortMADays,longMADays,aName);
  macdDecPar=[Note parName:parName];
  free(parName);

  if([self isInfoPresent:macdDecPar])
    return [self infoAsDouble:macdDecPar];
  else
    {
      // recursively find appropriate value
      double newDec = NO_SIGNAL;
```

```
    id previousNote=[[self part] previous:self];

    if(previousNote!=nil)
      {
        BOOL cont=YES;
        double prevDiff=[previousNote macdDiffForDays:signalDays andShortMA:shortMADay
s
                                    andLongMA:longMADays andParName:aName],
               thisDiff=[self macdDiffForDays:signalDays andShortMA:shortMADays
                                    andLongMA:longMADays andParName:aName];

        if(prevDiff * thisDiff < 0.0) /* MACD crosses signal line */
        {
            double thisMACD = [self macdWithShortMA:shortMADays andLongMA:longMADays a
ndParName:aName];
            double timeOfLastMACDZeroCrossing =
                                [self macdNegOf:thisMACD
                                forShortMA:shortMADays
                                andLongMA:longMADays
                                andParName:aName];
            double timeOfLastSignal=-DBL_MAX;

            while((cont==YES) && (previousNote!=nil)) /* find time of last signal */
            {
                if([previousNote macdDecisionForDays:signalDays
                            andShortMA:shortMADays
                            andLongMA:longMADays
                            andParName:aName]==NO_SIGNAL)
                    previousNote=[[self part] previous:previousNote];
                else
                {
                    timeOfLastSignal=[previousNote timeTag];
                    cont=NO;
                }
            }
            if(timeOfLastMACDZeroCrossing>timeOfLastSignal) /* valid signal */
            {
                if(thisDiff>0.0)
                { /* buy */
                    newDec=(double) BUY_SIGNAL;
                    /* TODO: check for positive divergence */
                }
                else
                { /* sell */
                    newDec= (double) SELL_SIGNAL;
                    /* TODO: check for negative divergence */
                }
            }
        }
    }
    // store appropriate value
    [self setPar:macdDecPar toDouble:newDec];
    return newDec;
  }
}

- (double)macdDecisionForDays:(unsigned int)signalDays
                    andShortMA:(unsigned int)shortMADays
                    andLongMA:(unsigned int)longMADays
                    andParName:(char *)aName
                    depreciated:(double)percentage
{
  double result;
  id previousNote=[[self part] previous:self];

  if((result=(double)[self macdDecisionForDays:signalDays
                            andShortMA:shortMADays
                            andLongMA:longMADays
                            andParName:aName])==NO_SIGNAL)
    result=depreciate([previousNote macdDecisionForDays:signalDays
                                    andShortMA:shortMADays
                                    andLongMA:longMADays
                                    andParName:aName
                                    depreciated:percentage],percentage);
  return result;
}

- (double)sumForParName:(char *)aName
{
  double sum=0.0;
  int rawPar=[Note parName:aName];
  int sumPar;
  char *sumName=(char *)malloc(80);

  sprintf(sumName,"sum_%s",aName);
  sumPar=[Note parName:sumName];
  free(sumName);
  if([self isInfoPresent:sumPar])
    sum=[self infoAsDouble:sumPar];
  else
    {
      id previousNote=[[self part] previous:self];

      if([self isInfoPresent:rawPar])
        sum=[self infoAsDouble:rawPar];
      if(previousNote!=nil)
        sum+=[previousNote sumForParName:aName];
      [self setPar:sumPar toDouble:sum];
    }
  return sum;
}

- (double)sumOfProductForParName:(char *)aName andParName:(char *)otherName
{
  double sum=0.0;
  int rawPar1=[Note parName:aName], rawPar2=[Note parName:otherName];
  int sumPar;
  char *sumName=(char *)malloc(160);

  sprintf(sumName,"sumProd_%s_%s",aName,otherName);
  sumPar=[Note parName:sumName];
  free(sumName);
  if([self isInfoPresent:sumPar])
    sum=[self infoAsDouble:sumPar];
  else
    {
      id previousNote=[[self part] previous:self];

      if([self isInfoPresent:rawPar1] && [self isInfoPresent:rawPar2])
        sum=[self infoAsDouble:rawPar1]*[self infoAsDouble:rawPar2];
      if(previousNote!=nil)
        sum+=[previousNote sumOfProductForParName:aName andParName:otherName];
      [self setPar:sumPar toDouble:sum];
    }
  return sum;
}

- (double)sumOfSquaresForParName:(char *)aName
```

```
{
  double sum=0.0;
  int rawPar=[Note parName:aName];
  int sumPar;
  char *sumName=(char *)malloc(160);

  sprintf(sumName,"sumSq_%s",aName);
  sumPar=[Note parName:sumName];
  free(sumName);
  if([self isInfoPresent:sumPar])
    sum=[self infoAsDouble:sumPar];
  else
    {
      id previousNote=[[self part] previous:self];

      if([self isInfoPresent:rawPar])
        {
          sum=[self infoAsDouble:rawPar];
          sum*=sum;
        }
      if(previousNote!=nil)
        sum+=[previousNote sumOfSquaresForParName:aName];
      [self setPar:sumPar toDouble:sum];
    }
  return sum;

}

- (double)sumForDays:(unsigned int)days andParName:(char *)aName
{
  id baseNote=[[self part] noteAtIndex:[[self part] indexOfNote:self]-days];

  return [self sumForParName:aName] - [baseNote sumForParName:aName];
}

- (double)sumOfProductForDays:(unsigned int)days andParName:(char *)aName andParName:(ch
ar *)otherName
{
  id baseNote=[[self part] noteAtIndex:[[self part] indexOfNote:self]-days];

  return [self sumOfProductForParName:aName andParName:otherName]-
         [baseNote sumOfProductForParName:aName andParName:otherName];
}

- (double)sumOfSquaresForDays:(unsigned int)days andParName:(char *)aName
{
  id baseNote=[[self part] noteAtIndex:[[self part] indexOfNote:self]-days];

  return [self sumOfSquaresForParName:aName]-
         [baseNote sumOfSquaresForParName:aName];
}

// TODO: presently, it is assumed that all data are pairwise available
- (double)linearRegressionCoefficientForDays:(unsigned int)days
                          andXPar:(char *)xPar andYPar:(char *)yPar
{
  double n;
  double sumX=[self sumForDays:days andParName:xPar];
  double sumY=[self sumForDays:days andParName:yPar];
  double sumXY=[self sumOfProductForDays:days andParName:xPar andParName:yPar];
  double sumSqX=[self sumOfSquaresForDays:days andParName:xPar];
  double coefficient;

  if((n=(double)[[self part] indexOfNote:self]+1.0)>days)
    n=(double)days;
  coefficient=(n*sumXY - sumX*sumY) / (n*sumSqX - sumX*sumX);
  return coefficient;
}

// TODO: in the current implementation, n assumes that all data are available
// up to the first day of available data
- (double)linearRegressionOffsetForDays:(unsigned int)days
                          andXPar:(char *)xPar andYPar:(char *)yPar
{
  double offset,n;
  //   (sumY - b*sumX)/n)
  double nominator=[self sumForDays:days andParName:yPar]
                   - [self linearRegressionCoefficientForDays:days andXPar:xPar andYPar
:yPar]
                     * [self sumForDays:days andParName:xPar];

  if((n=(double)[[self part] indexOfNote:self]+1.0)>days)
    n=(double)days;
  offset=nominator / n;
  return offset;
}

- ( int)marketType
{
  id previousNote=[[self part] previous:self];
  int lowPar=[Note parName:"low"];
  int highPar=[Note parName:"high"];
  double low=[self infoAsDouble:lowPar];
  double high=[self infoAsDouble:highPar];
  double prevLow=[previousNote infoAsDouble:lowPar];
  double prevHigh=[previousNote infoAsDouble:highPar];
   int marketType=0;

  if(previousNote==nil)
    marketType=0;
  else if((high>prevHigh) && (low>=prevLow ))
    marketType=RALLY;
  else if((low<prevLow) && (high<=prevHigh))
    marketType=REACTION;
  else if((high<=prevHigh) && (low>=prevLow))
    marketType=INSIDE;
  else if((high>prevHigh) && (low<prevLow))
    marketType=OUTSIDE;
  return marketType;
}

- ( int)marketTypeDays:(unsigned int)days
{
  id compNote=[[self part] noteAtIndex:[[self part] indexOfNote:self]-days];
   int marketType=0;

  if(compNote!=nil)
    {
      double low=[self minWithinDays:days forParName:"low"];
      double high=[self maxWithinDays:days forParName:"high"];
      double prevLow=[compNote minWithinDays:days forParName:"low"];
      double prevHigh=[compNote maxWithinDays:days forParName:"high"];

      if(compNote==nil)
        marketType=0;
      else if((high>prevHigh) && (low>=prevLow ))
        marketType=RALLY;
      else if((low<prevLow) && (high<=prevHigh))
```

```
        marketType=REACTION;
      else if((high<=prevHigh) && (low>=prevLow))
        marketType=INSIDE;
      else if((high>prevHigh) && (low<prevLow))
        marketType=OUTSIDE;
    }
  return marketType;
}


- ( int)increaseRelativeToNote:aNote forDays:(unsigned int)days forParName:(char *)aName
{
  double value=[self expMovingAverageForDays:days andParName:aName];
  double otherValue=[aNote expMovingAverageForDays:days andParName:aName];

  if(value>otherValue)
    return YES;
  else
    return NO;
}

- ( int)increaseRelativeToNote:aNote forParName:(char *)aName
{
  return [self increaseRelativeToNote:aNote forDays:1 forParName:aName];
}

- ( int)volumeReversalDays:(unsigned int)days
{
  id otherNote=self;
  int marketType=[self marketTypeDays:days];
  int otherMarketType;
  int ret=NO_SIGNAL;
  char *aName="volume";

  char *parName=(char *)malloc(80);
  int volRevPar;

  sprintf(parName,"volRev_%u",days);
  volRevPar=[Note parName:parName];
  free(parName);

  if([self isParPresent:volRevPar])
    return (int)[self infoAsDouble:volRevPar];
  else
  {
    do
      {
        otherNote=[[self part] previous:otherNote];
        otherMarketType=[otherNote marketTypeDays:days];
      }  while((otherNote!=nil) && ((otherMarketType==INSIDE) || (otherMarketType==OUTSI
DE)));
    if(otherNote!=nil)
      switch(otherMarketType)
        {
          case RALLY:    if(marketType==REACTION)
                           if([self increaseRelativeToNote:otherNote forDays:days forPar
Name:aName]==YES)
                             ret=NEG_VOL_REVERSAL;
                         break;
          case REACTION: if(marketType==RALLY)
                           if([self increaseRelativeToNote:otherNote forDays:days forPar
Name:aName]==YES)
                             ret=POS_VOL_REVERSAL;
                         break;
        }
    [self setPar:volRevPar toDouble:(double)ret];
    return ret;
  }
}

- (double)volumeReversalDays:(unsigned int)days depreciated:(double)percentage
{
  double result;
  id previousNote=[[self part] previous:self];

  if((result=(double)[self volumeReversalDays:days])==NO_SIGNAL)
    result=depreciate([previousNote volumeReversalDays:days depreciated:percentage],perce
ntage);
  return result;
}

- ( int)volumeReversal
{
  return [self volumeReversalDays:1];
}


- ( int)volumeReversalWeekly
{
  return [self volumeReversalDays:5];
}

- (int)weekday
{
  return weekday([self timeTag]);
}

- (int)year
{
  return year([self timeTag]);
}

- (int)dayOfYear
{
  return dayOfYear([self timeTag]);
}

- (int)santaClaus
{
  double tt=[self timeTag];
  int doy=dayOfYear(tt);

  if((doy>7) && (doy<355))
    return MIDDLE_DAY;
  else
    if(doy<=7)
      {
        int busDay=businessDayOfYear(tt);

        if(busDay<=2)
          return POS_INFLUENCE;
        else
          return NO_INFLUENCE;
      }
    else
      {
        int busDaysLeft=businessDaysLeftInYear(tt);
```

```
        if(busDaysLeft<=5)
          return POS_INFLUENCE;
        else
          return NO_INFLUENCE;
      }
}

- (int)santaClausYearInfluenceForParName:(char *)aName
{
  char *parName=(char *)malloc(80);
  int santaYrPar;

  sprintf(parName,"santaYr_%s",aName);
  santaYrPar=[Note parName:parName];
  free(parName);

  if([self isInfoPresent:santaYrPar])
    return [self infoAsDouble:santaYrPar];
  else
    {
      id previousNote=[[self part] previous:self];
      int parameter=[Note parName:aName];
      double newSantaYr=(double)NO_INFLUENCE;

      if(previousNote!=nil)
        {
          int thisBusinessDayOfYear=businessDayOfYear([self timeTag]);

          if(thisBusinessDayOfYear>2)
            // recursively accept prior entry
            newSantaYr=[previousNote santaClausYearInfluenceForParName:aName];
          else
            {
              unsigned int thisIndex=[[self part] indexOfNote:self];
              double thisValue=[self infoAsDouble:parameter];
              id compNote;

              if((compNote=[[self part] noteAtIndex:(thisIndex-5-(unsigned int)thisBusin
essDayOfYear)])!=nil)
                {
                  double compValue=[compNote infoAsDouble:parameter];
                  double ret=thisValue/compValue;

                  if(ret!=(double)1.0)
                    {
                      if(ret>(double)1.0)
                        newSantaYr=POS_INFLUENCE;
                      else
                        newSantaYr=NEG_INFLUENCE;
                    }
                }

            }
        }
      // store appropriate value
      [self setPar:santaYrPar toDouble:newSantaYr];
      return (int)newSantaYr;
    }
}

// temporary -- to be avoided
- ensureHighLowSpreadAvailableUpToNoteForDays:(unsigned int)days
{
  char *parName=(char *)malloc(80);
  int parHLSpread;

  sprintf(parName,"HLSpread_%u",days);
  parHLSpread=[Note parName:parName];
  free(parName);
  if([self isParPresent:parHLSpread]==NO)
      [self highLowSpread:days];
  [[[self part] previous:self] ensureHighLowSpreadAvailableUpToNoteForDays:days];
  return self;
}

/* although the high low spread is easily computed,
   we store it here because a variety of transformations
   of it is needed and thus, the transparency is
   advantageous.
 */
- (double)highLowSpread:(unsigned int)days
{
  char *parName=(char *)malloc(80);
  int parHLSpread;

  sprintf(parName,"HLSpread_%u",days);
  parHLSpread=[Note parName:parName];
  free(parName);

  if([self isInfoPresent:parHLSpread])
      return [self infoAsDouble:parHLSpread];
  else
  {
      double hlSpread=log([self maxWithinDays:days forParName:"high"] - [self minWithinDa
ys:days forParName:"low"]);

      [self setPar:parHLSpread toDouble:hlSpread];
      return hlSpread;
  }
}

@end
```

```
#import <musickit/musickit.h>
#import <objc/Object.h>
#import <musickit/Part.h>

@interface Part (CAFPart)
// no additional variables

// methods
- previous:aNote;
- (unsigned int)indexOfNote:aNote;
- noteAtIndex:(unsigned int)anIndex;
- atOrBeforeTime:(double)timeTag;

@end
```

```
#import "CAFPart.h"

@implementation Part (CAFPart)

- previous:aNote
{
  if(isSorted==NO)
    [self sort];
  return [notes objectAt: [notes indexOf:aNote] - 1];
}

- (unsigned int)indexOfNote:aNote
{
  return [notes indexOf:aNote];
}

- noteAtIndex:(unsigned int)anIndex
{
  return [notes objectAt:anIndex];
}

- atOrBeforeTime:(double)timeTag
{
  id afterNote=[self atOrAfterTime:timeTag];

  if([afterNote timeTag]==timeTag)
    return afterNote;
  else
    if(afterNote==nil)
      return [notes objectAt:[self noteCount]-1];
  else
    {
       unsigned int afterIndex=[notes indexOf:afterNote];

       if(afterIndex==0)
         return nil;
       else
         return [self noteAtIndex:(afterIndex-1)];
    }
}

// redundant since replaced with Note-methods infoAsDouble
//- timeTagAsParameter
//{
//  unsigned int count=[notes count],
//               x;
//  id aNote;
//  int timeTagPar=[Note parName:"timeTag"];
//  BOOL cont=YES;
//
//  if(![[notes objectAt:count-1] isParPresent:timeTagPar])
//    {
//      for(x=count;(x>0) && cont;x--)
//        {
//          aNote=[notes objectAt:x-1];
//          if([aNote isParPresent:timeTagPar])
//          cont=NO;
//        else
//            [aNote setPar:timeTagPar toDouble:[aNote timeTag]];
//        }
//    }
//  return self;
//}
```

```
@end
```

```
#import <objc/objc.h>
#import <time.h>
#import <math.h>
#import <stdlib.h>

#define SECONDS_PER_DAY 24*60*60      /* hours*minutes*seconds per day */
#define EST_ZONE -5

/* dates.h
 *
 * created 8 March 1991
 *
 * This is a generic time utility package that provides support for
 * time tags in data series. It tries to remain as compatible as possible
 * with the standard UNIX and VMS time formats, in that it counts the seconds
 * since 1970 to identify any point in time. Also all times should be
 * to UTC/GMT. However to allow a wider range of possible dates, we use
 * double floating point numbers that give us at least 4 times the range of the
 * integers used by the system built-in routines. Dates before 1970 are given in
 * form of negative numbers. As far as possible we also use the structs used in
 * time.h.
 *
 * To Do:
 *   - add support for the tzone support of the OS
 *   - add support for DOW (day of week)
 */

/*
 * returns the number of days that are correct
 * relative to the calendar valid for England and her colonies.
 * -> it may be used for date comparisons with any date
 * after September 1752 when the Gregorian calendar was introduced.
 */
int days(struct tm *tp);

/*
 * returns days since 1 Jan 1900 B.C. (post Christum natum)
 */
int daysSince1900(struct tm *tp);

/*
 * returns seconds since 1 Jan 1970, UTC/GMT,
 * negative numbers are dates before 1 Jan 1970
 */
double secondsSince1970(struct tm *tp);

/*
 * returns a time tag (i.e. seconds since 1 Jan 1970) for
 * a date specified by the respective integers.
 */
double timeTag(int year, int month, int day, int zone);

/* returns the day of the week (SUN=0, MON=1, ..., SAT=6)
 * for a specific time tag.
 */
int weekday(double timeTag);

/* returns the year of timeTag
 * plenty of room for optimization...
 */
int year(double timeTag);

int dayOfYear(double timeTag);

int businessDayOfYear(double timeTag);

int businessDaysLeftInYear(double timeTag);

void month_day(int year, int yearday, int *pmonth, int *pday);
```

```
#import "dates.h"

int weekday(double timeTag)
{
  static BOOL computed=NO;
  static int daysTo1970;
  if(computed==NO)
    {
      struct tm origin;
      (origin).tm_mday=1;
      (origin).tm_mon=0;
      (origin).tm_year=70;
      daysTo1970=days(&origin);
      computed=YES;
    }
  return (((int)(timeTag/((double)SECONDS_PER_DAY) + ((double)daysTo1970))) + 7) % 7;
}

int year(double aTimeTag)
{
  int startYear=(int)(aTimeTag/(((double)SECONDS_PER_DAY)*365.25))+70; /* 0==1970 */

  while(timeTag(startYear,1,1, EST_ZONE)<aTimeTag)
    {
      startYear++;
    }
  return --startYear;
}

int dayOfYear(double aTimeTag)
{
  double seconds;

  seconds=aTimeTag-timeTag(year(aTimeTag),1,1,EST_ZONE);

  return (int)floor(seconds/((double)SECONDS_PER_DAY))+1;

}

int businessDayOfYear(double aTimeTag)
{
  double otherTag=aTimeTag;
  int aYear=year(aTimeTag);
  int daysPast=0;
  int wd;
  int fd=1;
  double fdTag=timeTag(aYear,1,1,EST_ZONE);
  int wdfd=weekday(fdTag);

  if(wdfd==0)
    fd+=1;
  else
    if(wdfd==6)
      fd+=2;

  while(year(otherTag)==aYear)
    {
      wd=weekday(otherTag);
      if((wd>0) && (wd<6) && (dayOfYear(otherTag)!=fd))
        daysPast++;
      otherTag-=(double)SECONDS_PER_DAY;
    }
  return daysPast;
}

int businessDaysLeftInYear(double aTimeTag)
{
  double otherTag=aTimeTag;
  int aYear=year(aTimeTag);
  int daysLeft=0;
  double xmassTag=timeTag(aYear,12,25,EST_ZONE);
  int xmass=dayOfYear(xmassTag);
  int wdXmass=weekday(xmassTag);
  int wd;

  /* US holiday/weekend rule */
  if(wdXmass==0)
    xmass+=1;
  else
    if(wdXmass==6)
      xmass+=2;
//printf("x-dow:%i x-doy:%i ",wdXmass,xmass);

  while(year(otherTag)==aYear)
    {
      wd=weekday(otherTag);
      if((wd>0) && (wd<6)
         && (dayOfYear(otherTag)!=xmass))
           {
//printf("doy:%i ",dayOfYear(otherTag));
             daysLeft++;
           }
      otherTag+=(double)SECONDS_PER_DAY;
    }
  return daysLeft;
}

/************************************************/
int days(struct tm *tp)
{
  int total=0;
  int year= (*tp).tm_year+1900;
  unsigned int feb29s=(year>>2)+1;
  feb29s-= (year / 100 + 1);
  feb29s+= (year / 400);

  total+=year*365+(int)feb29s;
  if(((*tp).tm_mon<2) &&
     (year%4 == 0) && ((year%100!=0) || (year%400==0)))
    total--;
  switch((*tp).tm_mon)
    {
    case 0:    break;
    case 1:    total+=31;   break;
    case 2:    total+=59;   break;
    case 3:    total+=90;   break;
    case 4:    total+=120;  break;
    case 5:    total+=151;  break;
    case 6:    total+=181;  break;
    case 7:    total+=212;  break;
    case 8:    total+=243;  break;
    case 9:    total+=273;  break;
    case 10:   total+=304;  break;
    case 11:   total+=334;  break;
    }
  total+=(*tp).tm_mday;
  return total;
```

```
/*************************************************/
int daysSince1900(struct tm *tp)
{
  static BOOL computed=NO;
  static int daysTo1900;
  if(computed==NO)
    {
      struct tm origin;
      (origin).tm_mday=1;
      (origin).tm_mon=0;
      (origin).tm_year=0;          /* 1900 */
      daysTo1900=days(&origin);
      computed=YES;
    }
  return days(tp)-daysTo1900;
}


double secondsSince1970(struct tm *tp)
{
  static BOOL computed=NO;
  static int daysTo1970;
  if(computed==NO)
    {
      struct tm origin;
      (origin).tm_mday=1;
      (origin).tm_mon=0;
      (origin).tm_year=70;          /* 1970 */
      daysTo1970=days(&origin);
      computed=YES;
    }
  return (double)((days(tp)-daysTo1970)) * (double)SECONDS_PER_DAY + (double)(*tp).tm_gm
toff;
}


// uses system function for dates>1970, uses own function for other dates
// valid range: Jan 1, 1900 - Jan 1, 2038
double timeTag(int year,int month,int day,int zone)
{
  double timePoint;
  struct tm *tp=(struct tm *)calloc(1,sizeof(struct tm));

  (*tp).tm_mday=day;
  (*tp).tm_mon=month-1;
  (*tp).tm_year=year;
  (*tp).tm_gmtoff=zone*60*60; /* offset from GMT in seconds */

  // if one wanted to use mktime(tp), one would need to test if the date given
  // is within an acceptable range, i.e.
  //  if((*tp).tm_year<70 ||
  //     ((*tp).tm_year==70 && (*tp).tm_mon==0 && (*tp).tm_mday==1 && (*tp).tm_gmtoff<0)
}
  //   timePoint=(double)secondsSince1970(tp)
  //  else
  //    if((timePoint=mktime(tp))==-1)
  //      <error condition>

  timePoint=secondsSince1970(tp);
  free(tp);
  return timePoint;
}

void month_day(int year, int yearday, int *pmonth, int *pday)
{
  int i, leap;
  static char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

  leap = year % 4 == 0 && year % 1000 != 0 || year % 400 == 0;
  for(i=1; yearday > daytab[leap][i]; i++)
    yearday -= daytab[leap][i];
    *pmonth = i;
    *pday = yearday;
}
```

```
#import <nib/InterfaceBuilder.h>

@interface DensityButtonCellInspector:Inspector
{
    id        graySlider;
    id        grayTextField;
    id        disabledSwitch;
    id        tagTextField;
}

+ finishLoading:(struct mach_header *)header;
+ startUnloading;
- init;
- ok:sender;
- revert:sender;

@end
```

```
#import "DensityButtonCellInspector.h"
#import "DensityButtonCell.h"
#import <appkit/Application.h>
#import <appkit/Slider.h>
#import <appkit/TextField.h>

@implementation DensityButtonCellInspector

+ finishLoading:(struct mach_header *)header
{
    NIBDidLoadClass(self, header);
    return nil;
}

+ startUnloading
{
    NIBWillUnloadClass(self);
    return nil;
}

- init
{
    [super init];
    [NXApp loadNibSection:"DensityButtonCellInspector.nib" owner:self];
    return self;
}

- ok:sender
{
    [object setGrayValue:[graySlider floatValue]];
    [object setEnabled:(![disabledSwitch state])];
    [object setTag:[tagTextField intValue]];
    return [super ok:sender];
}

- revert:sender
{
    [graySlider setFloatValue:[object grayValue]];
    [grayTextField setFloatValue:[object grayValue]];
    [disabledSwitch setState:(![object isEnabled])];
    [tagTextField setIntValue:[object tag]];
    return [super revert:sender];
}

@end
```

```
#import <appkit/graphics.h>
#import <dpsclient/wraps.h>
#import <appkit/ButtonCell.h>

@interface DensityButtonCell:ButtonCell
{
    float grayValue;
}

- (id)setGrayValue:(float)aGrayValue;
- (float)grayValue;
- (id)drawInside:(const NXRect *)aRect inView:(id)controlView;
- takeGrayValueFrom:sender;
- (const char*)inspectorName;
- write:(NXTypedStream *)stream;
- read:(NXTypedStream *)stream;

@end
```

```
#import "DensityButtonCell.h"
#import <appkit/View.h>
#import <appkit/Control.h>

@implementation DensityButtonCell

- (id)setGrayValue:(float)aGrayValue
{
    grayValue=aGrayValue;
    return self;
}

- (float)grayValue
{
    return grayValue;
}

- (id)drawInside:(const NXRect *)aRect inView:(id)controlView
{
    [controlView lockFocus];
    PSsetgray([self grayValue]);
    NXRectFill(aRect);
    [controlView unlockFocus];
    return self;
}

- takeGrayValueFrom:sender
{
    if([sender respondsTo:@selector(grayValue)])
        [self setGrayValue:[sender grayValue]];
    else
        [self setGrayValue:[sender floatValue]];
    [[self controlView] updateCell:self];
    return self;
}

- (const char*)inspectorName
{
    return "DensityButtonCellInspector";
}

- write:(NXTypedStream *)stream
{
    [super write:stream];
    NXWriteType(stream,"f",&grayValue);
    return self;
}

- read:(NXTypedStream *)stream
{
    [super read:stream];
    NXReadType(stream,"f",&grayValue);
    return self;
}

@end
```

```
#import <appkit/Button.h>

@interface DensityButton:Button
{
}

//+ initialize; //see history in DensityButton.m
- init;
- initFrame:(const NXRect *)frameRect;
- initFrame:(const NXRect*)frameRect
  title:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag;
- initFrame:(const NXRect *)frameRect
  icon:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag;
- takeGrayValueFrom:sender;

@end
```

```
#import "DensityButton.h"
#import "DensityButtonCell.h"
#import <appkit/View.h>

@implementation DensityButton

// Here some interesting history:
// First what we wanted to do:
//+ initialize
//{
//        [super initialize];
//        [self setCellClass:[DensityButtonCell class]];
//        return self;
//}
//
// Now what it should be, since super is initialized automatically
// by the runtime system, if need be:
//+ initialize
//{
//    // to prevent interference with calls from subclasses:
//    if ( self == [DensityButton class] )
//    {
//        [self setCellClass:[DensityButtonCell class]];
//    }
//    return self;
//}
//
// Now, we realize that cellClass is not really a class variable, but a
// static variable, globally bound :-(
// So we have to set it temporarily during the init methods of the
// instances...
// Here is what we would like to do:
//- init
//{
//  id someCellClass;
//
//  someCellClass=[[DensityButton class] setCellClass:[DensityButtonCell class]];
//  [super init];
//  [[DensityButton class] setCellClass:someCellClass];
//  return self;
//}
//
// Now the last sad discovery: +setCellClass does not return the previous
// cellClass, thus we have to hard code as follows:

- init
{
    [[DensityButton class] setCellClass:[DensityButtonCell class]];
    [super init];
    [[DensityButton class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
{
    [[DensityButton class] setCellClass:[DensityButtonCell class]];
    [super initFrame:frameRect];
    [[DensityButton class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
  title:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag
{
    [[DensityButton class] setCellClass:[DensityButtonCell class]];
    [super initFrame:frameRect
          title:aString
          tag:anInt
          target:anObject
          action:aSelector
          key:charCode
          enabled: flag];
    [[DensityButton class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
  icon:(const char *)aString
  tag:(int)anInt
  target:anObject
  action:(SEL)aSelector
  key:(unsigned short)charCode
  enabled:(BOOL)flag
{
    [[DensityButton class] setCellClass:[DensityButtonCell class]];
    [super initFrame:frameRect
          icon:aString
          tag:anInt
          target:anObject
          action:aSelector
          key:charCode
          enabled: flag];
    [[DensityButton class] setCellClass:[ButtonCell class]];
    return self;
}

- takeGrayValueFrom:sender
{
//    if([sender respondsTo:@selector(grayValue)])
//      [cell setGrayValue:[sender grayValue]];
//    else
//      [cell setGrayValue:[sender floatValue]];
    [[self selectedCell] takeGrayValueFrom:sender];
    return [self display];
}

@end
```

```
#import <appkit/Matrix.h>

@interface DensityMatrix:Matrix
{
}

//+ initialize; //see history in DensityButton.m
- init;
- initFrame:(const NXRect *)frameRect;
- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  prototype:aCell
  numRows:(int)rowsHigh
  numCols:(int)colsWide;
- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  cellClass:factoryId
  numRows:(int)rowsHigh
  numCols:(int)colsWide;
- takeGrayValueFrom:sender;

@end
```

```
#import "DensityMatrix.h"
#import "DensityButtonCell.h"
#import <appkit/View.h>

@implementation DensityMatrix

// Here some interesting history:
// First what we wanted to do:
//+ initialize
//{
//        [super initialize];
//        [self setCellClass:[DensityButtonCell class]];
//        return self;
//}
//
// Now what it should be, since super is initialized automatically
// by the runtime system, if need be:
//+ initialize
//{
//    // to prevent interference with calls from subclasses:
//    if ( self == [DensityMatrix class] )
//    {
//        [self setCellClass:[DensityButtonCell class]];
//    }
//    return self;
//}
//
// Now, we realize that cellClass is not really a class variable, but a
// static variable, globally bound :-(
// So we have to set it temporarily during the init methods of the
// instances...
// Here is what we would like to do:
//- init
//{
//  id someCellClass;
//
//  someCellClass=[[DensityMatrix class] setCellClass:[DensityButtonCell class]];
//  [super init];
//  [[DensityMatrix class] setCellClass:someCellClass];
//  return self;
//}
//
// Now the last sad discovery: +setCellClass does not return the previous
// cellClass, thus we have to hard code as follows:

- init
{
#ifdef PALETTE
    id someCellPrototype;
#endif

    [[DensityMatrix class] setCellClass:[DensityButtonCell class]];
    [super init];
    [[DensityMatrix class] setCellClass:[ButtonCell class]];
#ifdef PALETTE
    someCellPrototype=[self setPrototype:[[DensityButtonCell allocFromZone:[self zone]]i
nit]];
    if(someCellPrototype!=nil)
        {
            //[[self prototype]setSize:[someCellPrototype size]];
            [someCellPrototype free];
        }
#endif
    return self;
}

- initFrame:(const NXRect *)frameRect
{
#ifdef PALETTE
    id someCellPrototype;
#endif

    [[DensityMatrix class] setCellClass:[DensityButtonCell class]];
    [super initFrame:frameRect];
    [[DensityMatrix class] setCellClass:[ButtonCell class]];
#ifdef PALETTE
    someCellPrototype=[self setPrototype:[[DensityButtonCell allocFromZone:[self zone]]in
it]];
    if(someCellPrototype!=nil)
        {
            //[[self prototype]setSize:[someCellPrototype size]];
            [someCellPrototype free];
        }
#endif
    return self;
}

- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  prototype:aCell
  numRows:(int)rowsHigh
  numCols:(int)colsWide
{
    [[DensityMatrix class] setCellClass:[DensityButtonCell class]];
    [super initFrame:frameRect
           mode:aMode
           prototype:aCell
           numRows:rowsHigh
           numCols:colsWide];
    [[DensityMatrix class] setCellClass:[ButtonCell class]];
    return self;
}

- initFrame:(const NXRect *)frameRect
  mode:(int)aMode
  cellClass:factoryId
  numRows:(int)rowsHigh
  numCols:(int)colsWide
{
    [[DensityMatrix class] setCellClass:[DensityButtonCell class]];
    [super initFrame:frameRect
           mode:aMode
           cellClass:factoryId
           numRows:rowsHigh
           numCols:colsWide];
    [[DensityMatrix class] setCellClass:[ButtonCell class]];
    return self;
}

- takeGrayValueFrom:sender
{
//    if([sender respondsTo:@selector(grayValue)])
//      [[self selectedCell] setGrayValue:[sender grayValue]];
//    else
//      [[self selectedCell] setGrayValue:[sender floatValue]];
    [[self selectedCell] takeGrayValueFrom:sender];
    return [self display];
}
```

```
@end
```

```
#import <objc/Object.h>
#import <appkit/Application.h>
#import <musickit/musickit.h>
#import <appkit/OpenPanel.h>
#import <appkit/Matrix.h>
#import <appkit/Button.h>
#import <appkit/Form.h>
#import <appkit/Panel.h>
#import "Actions.h"
#import "globals.h"
#import "dates.h"
#import "CAFPart.h"
#import "CAFNote.h"
#import "Statistics.h"
#import "StatisticsView.h"
#import "dates.h"

#define DAY_PAR 0
#define MONTH_PAR 1
#define YEAR_PAR 2
#define WEEKDAY_PAR 3

@interface DeTerminator:Object
{
    id inputIdentifiers; //matrix of button cells

    id openPanel,
       savePanel;
    id timeFrameWindow;
    id dateCurrentParameter;
    id dateFromParameter;
    id dateToParameter;
    int realTimeMode;

    id score;
    id part;
    id currentNote;

    id inspectedObject;
    id inspectorPanel;
    id binCountSlider;
    id binCountTextField;
}

- init;
- openRequest:sender;
- (int)openDocument:(const char *)fileName;
- saveRequest:sender;
- saveToRequest:sender;
- (int)saveDocument:(const char *)fileName;
- (int)saveToDocument:(const char *)fileName;

- animateUpdateForTimeFrame:sender;
- updateForTime:sender;
- forceUpdate:sender;
-(id) inputUpdated;
-(id) batchModeUpdated;
- currentEntry;
- lastEntry;

- timeFrameWindowMakeKeyAndOrderFront:sender;
- (double)fromTime;
- (double)toTime;
- currentTime;
-inspect:sender;
-ok:sender;
-revert:sender;

- showError: (const char *)errorMessage;
- free;

@end
```

```
#import "DeTerminator.h"

@implementation DeTerminator

- init
{
  [super init];
  [NXApp setDelegate:self];

  openPanel = [OpenPanel new];
  savePanel = [SavePanel new];
  score=[[Score alloc] init];
  part=nil;
  currentNote=nil;
  realTimeMode=NO;

  return self;
}

- appDidInit:sender
{
  [NXApp loadNibSection:"TimeFrame.nib" owner:self];
  return self;
}

- openRequest:sender
{
  const char *fileName;
  // TODO: add string table support
  //  const char *const types[2] = ([stringSet valueForStringKey:"extension"],
  //                                NULL);
  const char *const types[3] = ("score","playscore", NULL);

  if ([openPanel runModalForTypes:types] && (fileName = [openPanel filename]))
    [self openDocument:fileName];
  else
    // TODO: add string table support
    //    [self showError:[stringSet valueForStringKey:"errorOnOpenRequest"]];
    [self showError:"errorOnOpenRequest"];

  return self;
}

- (int)openDocument:(const char *)fileName
{
  char *aFileName=(char *)calloc(512,sizeof(char));

  strcpy(aFileName,fileName);
  if(part!=nil)
    [[score freeParts] freeNotes];
  if([score readScorefile:aFileName]==nil)
    {
      [self showError:"errorOnFileRead"];
      return -1;
    }
  part=[[score parts] objectAt:0];
  currentNote=[[part sort] nth:0];
  free(aFileName);
  return 0;
}

- saveRequest:sender
{
  const char *fileName;
  const char *const types[2] = ("score", NULL);

  [savePanel setRequiredFileType:types[0]];
  if(score==nil)
    [self showError:"errorOnSaveRequest"];
  else
    {
      if([savePanel runModal] && (fileName=[savePanel filename]))
        [self saveDocument:fileName];
      else
        [self showError:"errorOnSaveRequest"];
    }
  return self;
}

- saveToRequest:sender
{
  const char *fileName;
  const char *const types[2] = ("playscore", NULL);

  [savePanel setRequiredFileType:types[0]];
  if(score==nil)
    [self showError:"errorOnSaveRequest"];
  else
    {
      if([savePanel runModal] && (fileName=[savePanel filename]))
        [self saveToDocument:fileName];
      else
        [self showError:"errorOnSaveRequest"];
    }
  return self;
}

- (int)saveDocument:(const char *)fileName
{
  if([score writeScorefile:(char *)fileName]==nil)
    {
      [self showError:"errorOnFileWrite"];
      return -1;
    }
  return 0;
}

- (int)saveToDocument:(const char *)fileName
{
  if([score writeOptimizedScorefile:(char *)fileName]==nil)
    {
      [self showError:"errorOnFileWrite"];
      return -1;
    }
  return 0;
}

- animateUpdateForTimeFrame:sender
{
  double fromTime=[self fromTime];
  double toTime=[self toTime] + ((double)(SECONDS_PER_DAY - 1));

  id noteList=[part firstTimeTag:fromTime lastTimeTag:toTime];
  unsigned int noteCount=[noteList count];
  int x=0;

  for(x=0;currentNote=[noteList objectAt:x],x<noteCount;x++)
    [self forceUpdate:sender];
```

```
  [noteList free];
  return self;

}

- updateForTime:sender
{
  double toTime=[self toTime] + ((double)(SECONDS_PER_DAY - 1));
  id showNote=[part atOrBeforeTime:toTime];

  if(showNote!=nil)
    {
      currentNote=showNote;
      [self forceUpdate:sender];
    }
  return self;
}

- forceUpdate:sender
{
    [self currentTime];
    [self inputUpdated];
    if(realTimeMode==NO)
      [self batchModeUpdated];
    return self;
}

-(id) inputUpdated
{
    unsigned int i;
    int numCells=[inputIdentifiers cellCount];

    for(i=0;i<numCells;i++)
        {
            [[[inputIdentifiers cellList]objectAt:i]performClick:self];
            [[[inputIdentifiers cellList]objectAt:i]propagateChange:self];
            [[[inputIdentifiers cellList]objectAt:i]performClick:self];
        }
    return self;
}

-(id) batchModeUpdated
{
    unsigned int i;
    int numCells=[inputIdentifiers cellCount];

    for(i=0;i<numCells;i++)
        {
            [[[inputIdentifiers cellList]objectAt:i]performClick:self];
            [[[inputIdentifiers cellList]objectAt:i]batchMode:self];
            [[[inputIdentifiers cellList]objectAt:i]performClick:self];
        }
  return self;
}


- currentEntry
{
  return currentNote;
}

- lastEntry
{
  return [[part notes] lastObject];
}

- timeFrameWindowMakeKeyAndOrderFront:sender
{
 return [timeFrameWindow makeKeyAndOrderFront:sender];
}

- (double)fromTime
{
  return timeTag([dateFromParameter intValueAt:YEAR_PAR],
                 [dateFromParameter intValueAt:MONTH_PAR],
                 [dateFromParameter intValueAt:DAY_PAR],
                 EST_ZONE);
}

- (double)toTime
{
  return timeTag([dateToParameter intValueAt:YEAR_PAR],
                 [dateToParameter intValueAt:MONTH_PAR],
                 [dateToParameter intValueAt:DAY_PAR],
                 EST_ZONE);
}

- currentTime
{

    double theTag=[currentNote timeTag];
    int month, day, theYear=year(theTag);
    const char *weekdayName[7]={"Sunday","Monday","Tuesday","Wednesday",
                                "Thursday","Friday","Saturday"};

    month_day(theYear,dayOfYear(theTag),&month,&day);

  printf("%s %i/%i/%i\n",weekdayName[weekday(theTag)],month,day,theYear);
  [dateCurrentParameter setIntValue:theYear at:YEAR_PAR];
  [dateCurrentParameter setIntValue:month at:MONTH_PAR];
  [dateCurrentParameter setIntValue:day at:DAY_PAR];
  [dateCurrentParameter setStringValue:weekdayName[weekday(theTag)] at:WEEKDAY_PAR];
  return self;
}

-inspect:sender
{
    inspectedObject=[[sender superview]superview];
    [self revert:self];
    [inspectorPanel makeKeyAndOrderFront:self];
    return self;
}

-ok:sender
{
    if(inspectedObject==nil)
        return self;
    else
        {
            [[inspectedObject statistics]setBinCount:[binCountSlider intValue]];
            [inspectedObject update:[inspectedObject statistics]];
            return self;
        };
}

- realTimeMode:sender
{
  realTimeMode= !realTimeMode;
```

```
  return self;
}

-revert:sender
{
    if(inspectedObject==nil)
        return self;
    else
        {
            [binCountSlider setIntValue:[[inspectedObject statistics]binCount]];
            [binCountTextField setIntValue:[[inspectedObject statistics]binCount]];
            return self;
        };
}

- showError: (const char *)errorMessage
{
  NXRunAlertPanel(NULL, errorMessage,"OK", NULL, NULL);
  // TODO: add string table support
  //  NXRunAlertPanel(NULL, errorMessage,
  //              [stringSet valueForStringKey:"OK"], NULL, NULL);
  return self;
}

- free
{
  [openPanel free];
  [savePanel free];
  [score free];
  return [super free];
}

@end
```

```
/*
 *     Generated by the NeXT Interface Builder.
 */

#import <stdlib.h>
#import <appkit/Application.h>

void main(int argc, char *argv[]) {
    NXApp = [Application new];
    [NXApp loadNibSection:"DeTerminator.nib" owner:NXApp];
    [NXApp run];
    [NXApp free];
    exit(0);
}
```

```
#import "Rule.h"


@interface Rule (DistributionFunctions)
// no new instance variables in category

// additional methods
- independentDistWithInput:(const double *)valueX
                           :(const double *)probX
                           :(const double *)valueY
                           :(const double *)probY
                           andOutput:(double *)output;

@end
```

```
#import <objc/Object.h>
#import <appkit/Application.h>
#import "globals.h"
#import "Input.h"
#import "DeTerminator.h"
#import "CAFNote.h"

#define STDDEV_GENERAL 0.1

@interface Input (InputInitFunctions)
// no new instance variables in category

// additional methods
- zeroMeanForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- shortLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- longLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- fastMacdForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- slowMacdForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- volumeForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- longVolumeForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- onBalanceVolumeForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- volumeReversalSignalForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- volumeReversalPeriodForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- weeklyVolumeReversalSignalForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- weeklyVolumeReversalPeriodForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- volumeShortLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- volumeLongLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- highLowSpreadForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- weeklyHighLowSpreadForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- highLowSpreadShortLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- highLowSpreadLongLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- santaForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- santaYearInfluenceForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- mondayForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- electionCycleForStatistics:someStatistics
                    andDistribution:(const char *)distributionName;
- hussein:someStatistics
                    andDistribution:(const char *)distributionName;
@end
```

```
#import "InputInitFunctions.h"
#import "preprocessing.h"

@implementation Input (InputInitFunctions)

- zeroMeanForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=0.0;

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- shortLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double shortLinearRegr=[[[NXApp delegate] currentEntry]
                          linearRegressionCoefficientForDays:5
                                  andXPar:"index"
                                  andYPar:"close"];
  const double stddevShortLinearRegr=15.98;
  double mean=sigmoidLinearLowToHigh(shortLinearRegr,0.0,1.0/(2.0*stddevShortLinearRegr)
);

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- longLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double longLinearRegr=[[[NXApp delegate] currentEntry]
                          linearRegressionCoefficientForDays:50
                                  andXPar:"index"
                                  andYPar:"close"];
  const double stddevLongLinearRegr=5.29;
  double mean=sigmoidLinearLowToHigh(longLinearRegr,0.0,1.0/(2.0*stddevLongLinearRegr));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- fastMacdForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry]
                      macdDecisionForDays:3
                      andShortMA:6
                      andLongMA:19
                      andParName:"close"
                      depreciated:0.05];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- slowMacdForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry]
                      macdDecisionForDays:9
                      andShortMA:19
                      andLongMA:39
                      andParName:"close"
                      depreciated:0.05];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- volumeForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double volume=[[[NXApp delegate] currentEntry] expMovingAverageForDays:3 andParName:"vo
lume"];
  const double avgVolume=168.0;
  const double stddevVolume=34.2;
  double mean=sigmoidLinearLowToHigh((volume/1000000.0)-avgVolume,0,1.0/(2.0*stddevVolume
));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- longVolumeForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double volume=[[[NXApp delegate] currentEntry] expMovingAverageForDays:20 andParName:"v
olume"];
  const double avgLongVolume=168.0;
  const double stddevLongVolume=22.2;
  double mean=sigmoidLinearLowToHigh((volume/1000000.0)-avgLongVolume,0,1.0/(2.0*stddevLo
ngVolume));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- onBalanceVolumeForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double obv=[[[NXApp delegate] currentEntry] onBalanceVolumeForParName:"close"]/10000000
0;
```

```
  double mean=sigmoidLinearLowToHigh(obv,-.67,1.0/(2.0*2.910140564));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- volumeReversalSignalForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry] volumeReversalDays:1 depreciated:0.40];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- volumeReversalPeriodForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry] volumeReversalDays:1 depreciated:0.05];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}


- weeklyVolumeReversalSignalForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry] volumeReversalDays:5 depreciated:0.40];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- weeklyVolumeReversalPeriodForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry] volumeReversalDays:5 depreciated:0.05];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- volumeShortLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double volShortLinRegr=[[[NXApp delegate] currentEntry]
                          linearRegressionCoefficientForDays:3
                                        andXPar:"index"
                                        andYPar:"volume"];
  const double stddevVolShortLinRegr=22.8;
  double mean=sigmoidLinearLowToHigh(volShortLinRegr/1000000.0,0.0,1.0/(2.0*stddevVolShor
tLinRegr));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- volumeLongLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double volLongLinRegr=[[[NXApp delegate] currentEntry]
                          linearRegressionCoefficientForDays:20
                                        andXPar:"index"
                                        andYPar:"volume"];
  const double stddevVolLongLinRegr=2.7;
  double mean=sigmoidLinearLowToHigh(volLongLinRegr/1000000.0,0.0,1.0/(2.0*stddevVolLongL
inRegr));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- highLowSpreadForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev = STDDEV_GENERAL;
  const double avgHighLowSpread = 4.26;
  const double stddevHighLowSpread = 0.405;
  double mean = sigmoidLinearLowToHigh([[[NXApp delegate] currentEntry]
                                        highLowSpread:3]-avgHighLowSpread,0.0,1.0/(2.0*s
tddevHighLowSpread));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- weeklyHighLowSpreadForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev = STDDEV_GENERAL;
  const double avgWeeklyHighLowSpread = 4.84;
  const double stddevWeeklyHighLowSpread = 0.404;
  double mean = sigmoidLinearLowToHigh([[[NXApp delegate] currentEntry]
                                        highLowSpread:10]-avgWeeklyHighLowSpread,
                                       0.0,
                                       1.0/(2.0*stddevWeeklyHighLowSpread));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- highLowSpreadShortLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
```

```
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev = STDDEV_GENERAL;
  const double stddevHighLowSpreadShortLin = 0.19327;
  double mean, hl;
  static id lastEntry=nil;

  if(lastEntry!=[[NXApp delegate] lastEntry])
    lastEntry=[[[NXApp delegate] lastEntry] ensureHighLowSpreadAvailableUpToNoteForDays:
3];
  hl=[[[NXApp delegate] currentEntry] linearRegressionCoefficientForDays:3 andXPar:"inde
x" andYPar:"HLSpread_3"];
  mean = sigmoidLinearLowToHigh(hl,0.0,1.0/(2.0*stddevHighLowSpreadShortLin));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- highLowSpreadLongLinearRegressionForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev = STDDEV_GENERAL;
  const double stddevHighLowSpreadLongLin = 0.04474;
  double mean, hl;
  static id lastEntry=nil;

  if(lastEntry!=[[NXApp delegate] lastEntry])
    lastEntry=[[[NXApp delegate] lastEntry] ensureHighLowSpreadAvailableUpToNoteForDays:
10];

  hl=[[[NXApp delegate] currentEntry] linearRegressionCoefficientForDays:20 andXPar:"ind
ex" andYPar:"HLSpread_10"];
  mean = sigmoidLinearLowToHigh(hl,0.0,1.0/(2.0*stddevHighLowSpreadLongLin));

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- santaForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=[[[NXApp delegate] currentEntry] santaClaus];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- santaYearInfluenceForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribu
tionName)];
  double stddev= STDDEV_GENERAL;
  double mean=(double)[[[NXApp delegate] currentEntry] santaClausYearInfluenceForParName
:"close"];

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- mondayForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double mean=linearLowToHigh((double)[[[NXApp delegate] currentEntry] weekday],(double)S
UNDAY,(double)SATURDAY);

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- electionCycleForStatistics:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double mean=linearLowToHigh((double)([[[NXApp delegate] currentEntry] year] % 4),0.0,3.
0);

  distInitFunction(someStatistics, sel_getUid((STR)distributionName), mean, stddev);
  return self;
}

- hussein:someStatistics
                    andDistribution:(const char *)distributionName
{
  DIST_IMP distInitFunction=(DIST_IMP)[someStatistics methodFor:sel_getUid((STR)distribut
ionName)];
  double stddev= STDDEV_GENERAL;
  double biModality=0.8;

  double fromTime=timeTag(91,1,13,EST_ZONE), toTime=timeTag(91,1,21,EST_ZONE);
  double currentTime=[[[NXApp delegate] currentEntry] timeTag];

  if((currentTime<fromTime) || (currentTime>toTime))
    [self zeroMeanForStatistics:someStatistics andDistribution:"fillWithMean:stdDev:"];
  else
    {
      distInitFunction(someStatistics, sel_getUid((STR)distributionName), biModality, std
dev);
    }
  return self;
}


@end
```

```
#import "StatisticsView.h"

typedef id (*INPUT_IMP)(id, SEL, id, const char*);
typedef id (*DIST_IMP)(id, SEL, double, double);

@interface Input:StatisticsView
{
    id dependentRulesIdentifiers; //matrix of Buttons
    id inputInitFunctionNameField;
    id distributionInitFunctionNameField;
}

- forceUpdate:sender;
- performClick:sender;
-(BOOL)dependentRules;
-(id) outputUpdated;
-(id) batchModeUpdated;
- propagateChange:sender;
- batchMode:sender;
- batchMode:sender;
-(BOOL)isProcessedChangeSignificant:sender;
-(id) write:(NXTypedStream *)stream;
-(id) read:(NXTypedStream *)stream;
//-(id)awake;

@end
```

```
#import "Input.h"
#import "Statistics.h"
#import <appkit/Matrix.h>
#import <objc/List.h>
#import <stdlib.h>

@implementation Input

- forceUpdate:sender
{
    return [self propagateChange:self];
}

- performClick:sender
{
    return self;
}

-(BOOL)dependentRules
{
    unsigned int i=0;
    int numCells=[dependentRulesIdentifiers cellCount];
    BOOL dependentRules=NO;

    while((dependentRules==NO) && (i<numCells))
        {
            if([[[dependentRulesIdentifiers cellList] objectAt:i] target]!=nil)
              dependentRules=YES;
            i++;
        }
  return dependentRules;
}


-(id) outputUpdated
{
    unsigned int i;
    int numCells=[dependentRulesIdentifiers cellCount];

    [self update:[self statistics]];
    for(i=0;i<numCells;i++)
        {
            [[[dependentRulesIdentifiers cellList]objectAt:i]performClick:self];
            [[[dependentRulesIdentifiers cellList]objectAt:i]propagateChange:self];
            [[[dependentRulesIdentifiers cellList]objectAt:i]performClick:self];
        }
    return self;
}

-(id) batchModeUpdated
{
    unsigned int i;
    int numCells=[dependentRulesIdentifiers cellCount];

    if([self dependentRules]==NO)
      {
        printf("%s \tmean %f \tstddev %f\n",[self title],[[self statistics] mean],[[sel
f statistics] standardDeviation]);
      }
    else
      {
        for(i=0;i<numCells;i++)
        {
            [[[dependentRulesIdentifiers cellList]objectAt:i]performClick:self];
            [[[dependentRulesIdentifiers cellList]objectAt:i]batchMode:self];
            [[[dependentRulesIdentifiers cellList]objectAt:i]performClick:self];
        }
      }
    return self;
}

- propagateChange:sender
{
    if([self isProcessedChangeSignificant:sender])
        {
            //if there are significant changes, promote them
            [self outputUpdated];
        }
    return self;
}

- batchMode:sender
{
//    if([self isProcessedChangeSignificant:sender])
//      {
//          //if there are significant changes, promote them
//          [self batchModeUpdated];
//      }
    [self batchModeUpdated];
    return self;
}

-(BOOL)isProcessedChangeSignificant:sender
{
    SEL inputInitFunctionSelector=sel_getUid((STR)[inputInitFunctionNameField stringValue
]);
    INPUT_IMP inputInitFunction;

    //test if the selectors are valid
    if(!([self respondsTo:inputInitFunctionSelector]
         &&
         [[Statistics class] instancesRespondTo:sel_getUid((STR)[distributionInitFunction
NameField stringValue])]))
        {
            char *tempString=NXZoneCalloc([self zone], 1023, sizeof(char));

            sprintf(tempString,"Either your Input Init Function (IIF) and/or \nDistributi
on Init Function (DIF) \nis/are invalid in %s.\nCheck spelling!", [self title]);
            [self showError:tempString];
            NXZoneFree([self zone], tempString);
            return NO;
        }
    inputInitFunction=(INPUT_IMP)[self methodFor:inputInitFunctionSelector];
    //do updates
    inputInitFunction(self, inputInitFunctionSelector,[self statistics],(const char *)[di
stributionInitFunctionNameField stringValue]);

    //test for significance for speed up
    //can use variable in future implementations to have different
    //speed vs. precision trade-offs
    //return if changes were significant
    //YES for the time being
    return YES;
}

-(id) write:(NXTypedStream *)stream
```

```
    [super write:stream];
    NXWriteObjectReference(stream, dependentRulesIdentifiers);
    NXWriteObjectReference(stream, inputInitFunctionNameField);
    NXWriteObjectReference(stream, distributionInitFunctionNameField);
    return self;
}

-(id) read:(NXTypedStream *)stream
{
    [super read:stream];
    dependentRulesIdentifiers = NXReadObject(stream);
    inputInitFunctionNameField = NXReadObject(stream);
    distributionInitFunctionNameField = NXReadObject(stream);
    return self;
}

//-(id)awake
//{
//    [super awake];
//    return self;
//}

@end
```

```
#import "globals.h"

double linearLowToHigh(double value, double minValue, double maxValue);
double sigmoid(double value, double threshold, double slope);
double sigmoidLinearLowToHigh(double value,double height,double slope);
double depreciate(double prevSignal, double percentage);
```

```
#import "preprocessing.h"
#import <math.h>

double linearLowToHigh(double value, double minValue, double maxValue)
{
  const double low=(double)LOW;
  const double high=(double)HIGH;

  return (value - minValue) / (maxValue - minValue) * (high - low) + low;
}

double sigmoid(double value, double threshold, double slope)
{
  return 1.0/(1.0+exp(-(value*slope*4.0-threshold*slope*4.0)));
}

/* results in -1 -> +1 output */
double sigmoidLinearLowToHigh(double value,double height,double slope)
{
  return sigmoid(value,(-height)  *(1.0/slope),slope)*((double)(HIGH - LOW))+(double)LOW
;
}

double depreciate(double prevSignal, double percentage)
{
  double mean=((double)HIGH) - ((double)(HIGH - LOW))/2.0;

  return (mean - prevSignal) * percentage + prevSignal;
}
```

```
#import <appkit/nextstd.h> /* MAX defined */
#import "Rule.h"
#import "globals.h"

#define MONDAY_SIGNIFICANCE 0.1
#define ELECTION_SIGNIFICANCE 0.1
#define FAST_MACD_SIGNIFICANCE 0.3

@interface Rule (RuleFunctions)
// no new instance variables in category

// additional methods
- identical:(double *)inputA withReserved:(double *)inputB andOutput:(double *)output;
- negate:(double *)inputA withReserved:(double *)inputB andOutput:(double *)output;
- absolute:(double *)inputA withReserved:(double *)inputB andOutput:(double *)output;
- heighten:(double *)inputA by:(double *)percentage andOutput:(double *)output;
- lower:(double *)inputA by:(double *)percentage andOutput:(double *)output;
- adjust:(double *)inputA by:(double *)inputB andOutput:(double *)output;
- mondayWithInput:(double *)inputA andWeekday:(double *)squashedWeekday andOutput:(doubl
e *)output;
- electionCycleWithInput:(double *)inputA andElectionYear:(double *)squashedElectionYear
 andOutput:(double *)output;
- weightEquallyWithInputs:(double *)inputA :(double *)inputB andOutput:(double *)output;
- weightTwoThirdsInput:(double *)inputA withOneThirdInput:(double *)inputB andOutput:(do
uble *)output;
- weightHighlySignificantInput:(double *)inputA withLessSignificantInput:(double *)input
B andOutput:(double *)output;
- weightEquallyAndNegateWithInputs:(double *)inputA :(double *)inputB andOutput:(double
*)output;
- absoluteDifferenceOfInputs:(double *)inputA :(double *)inputB andOutput:(double *)outp
ut;

@end
```

```
#import "RuleFunctions.h"
#import "preprocessing.h"

@implementation Rule (RuleFunctions)

- identical:(double *)inputA withReserved:(double *)inputB andOutput:(double *)output
{
  // probability of reserved should be uniformly distributed
  // for LDW inputA=normVolShortLinRegr
  //         normVolShortLinRegr=sigmoidLinearLowToHigh(volShortLinRegr/1000000.0,0,1.0/
(2.0*22.8))
  //         volShortLinRegr=[aNote linearRegressionCoefficientForDays:3 andXPar:"index"
 andYPar:"volume"])
  //
  // for LDC inputA=normalizedVolLongLinRegr
  //         normVolLongLinRegr=sigmoidLinearLowToHigh(volLongLinRegr/1000000.0,0,1.0/(2
.0*2.7))
  //         volShortLinRegr=[aNote linearRegressionCoefficientForDays:20 andXPar:"index
" andYPar:"volume"])
  //
  // for VSW inputA=normHLSpread
  //         normHLSpread=sigmoidLinearLowToHigh([aNote highLowSpread:3]-4.26,0.0,1.0/(2
.0*0.405))
  //
  // for VSC
  //         normWeeklyHLSpread=sigmoidLinearLowToHigh([aNote highLowSpread:10]-4.84,0.0
,1.0/(2.0*0.404))
  //
  // for VDW
  //         normHLSpreadShortLin=sigmoidLinearLowToHigh(HLSpreadShortLin,1.0/(2.0*0.193
27))
  //         HLSpreadShortLin=[aNote linearRegressionCoefficientForDays:3 andXPar:"index
" andYPar:"HLSpread_3"])
  //         must ensure that parameter HLSpread_3 exists: [lastNote ensureHighLowSpread
AvailableUpToNoteForDays:3]
  //
  // for VDC
  //         normHLSpreadLongLin=sigmoidLinearLowToHigh(HLSpreadLongLin,1.0/(2.0*0.04474
))
  //         HLSpreadLongLin=[aNote linearRegressionCoefficientForDays:20 andXPar:"index
" andYPar:"HLSpread_10"])
  //         must ensure that parameter HLSpread_3 exists: [lastNote ensureHighLowSpread
AvailableUpToNoteForDays:10]


  (*output) = *inputA;
  return self;
}

- negate:(double *)inputA withReserved:(double *)inputB andOutput:(double *)output
{
  // probability of reserved should be uniformly distributed
  // for PDW.1, PDC.1: inputA=OBV
  // OBVa=[aNote onBalanceVolumeForParName:"close"]/100000000.0);
  // OBV = sigmoidLinearLowToHigh( OBVa - 12.27, 0.0, 1.0/(2.0*9.101))

  (*output)=((*inputA)
                  - (double)LOW - ((double)(HIGH - LOW))/2.0)
                  * (-1.0)
                  + ((double)(HIGH - LOW))/2.0 + (double)LOW;
  return self;
}

- absolute:(double *)inputA withReserved:(double *)inputB andOutput:(double *)output
{
  // probability of reserved should be uniformly distributed
  // for LSW.1 inputA=volumeReversalHD
  // volumeReversalHD=depreciate(prevVolRev,0.40)  // high depreciation
  //           thisVolumeReversal=[aNote volumeReversalDays:1]
  //
  // for LSC.1 inputA=weeklyVolumeReversalHD
  // weeklyVolumeReversalHD=depreciate(prevWeeklyVolRev,0.40)  // high depreciation
  //           thisWeeklyVolumeReversal=[aNote volumeReversalDays:5]

  if((*inputA) < ((double)(HIGH - LOW))/2.0)
    [self negate:inputA withReserved:inputB andOutput:output];
  return self;
}

// for now, we will have very simple rules that we expand upon having the
// framework set up.
- heighten:(double *)inputA by:(double *)percentage andOutput:(double *)output
{
  const double high=(double)HIGH;

  (*output) = (high- (*inputA)) * (*percentage)+(*inputA);
  if((*output)> HIGH )
    (*output)= HIGH ;
  return self;
}

- lower:(double *)inputA by:(double *)percentage andOutput:(double *)output
{
  const double low=(double)LOW;

  (*output) = (low-(*inputA)) * (*percentage) + (*inputA);
  if((*output)< LOW )
    (*output)= LOW;
  return self;
}

- adjust:(double *)inputA by:(double *)inputB andOutput:(double *)output
{
  // for PSW: inputA=shortLinear, inputB=santa
  // shortLinear ranges from low to high
  // shortLinear=sigmoidLinearLowToHigh([aNote linearRegressionCoefficientForDays:5
  //                                                    andXPar:"index"
  //                                                    andYPar:"close"],
  //                                              0.0,1.0/(2.0*15.98));
  // santa, when positive should be about 0.3
  // santa, when no influence should be about 0
  // santa=[aNote santaClaus]
  //
  // for PSC.3: inputA=output of PSC.2 inputB=santaYearly
  // santaYearly=[aNote santaClausYearInfluenceForParName:"close"],
  //
  // for PDW.2: inputA=PDW.1 inputB=factmacd
  // fastmacd=depreciate(prevFactMacd,0.05)
  //          thisMacd=[aNote
  //                    macdDecisionForDays:3
  //                    andShortMA:6
  //                    andLongMA:19
  //                    andParName:"close"];
  //
  // for PDW.3: inputA=PDW.2 inputB=volumeReversal
  // volumeReversal=depreciate(prevVolRev,0.05)
  //           thisVolumeReversal=[aNote volumeReversalDays:1]
  // for PDW.4: inputA=PDW.3 inputB=santa
```

```
  //
  // for PDC.2: inputA=PDC.1 inputB=slowmacd
  // slowmacd=depreciate(prevSlowMacd,0.05)
  //          thisMacd=[aNote
  //                    macdDecisionForDays:9
  //                    andShortMA:19
  //                    andLongMA:39
  //                    andParName:"close"];
  //
  // for PDC.3: inputA=PDC.2 inputB=weeklyVolumeReversal
  // weeklyVolumeReversal=depreciate(prevWeeklyVolRev,0.05)
  //           thisWeeklyVolumeReversal=[aNote volumeReversalDays:5]
  //
  // for LSW.2: inputA=volume inputB=LSW.1
  // volume=sigmoidLinearLowToHigh((volume3d/1000000.0)-168.0,0,1.0/(2.0*34.2))
  //           volume3d=[aNote expMovingAverageForDays:3 andParName:"volume"]
  //
  // for LSC.2: inputA=volume inputB=LSC.1
  // volume=sigmoidLinearLowToHigh((volume20d/1000000.0)-168.0,0,1.0/(2.0*22.2))
  //           volume20d=[aNote expMovingAverageForDays:20 andParName:"volume"]
  const double mean=   ((double)HIGH) - ((double)(HIGH - LOW))/2.0;
  double negB= -(*inputB);

  if((*inputB)>= mean)
    [self heighten:inputA by:inputB andOutput:output];
  else
    [self lower:inputA by:&negB andOutput:output];
  return self;
}

- mondayWithInput:(double *)inputA andWeekday:(double *)squashedWeekday andOutput:(doubl
e *)output
{
  // for PSC.1: inputA=longLinear
  // longLinear=sigmoidLinearLowToHigh([aNote linearRegressionCoefficientForDays:50
  //                                                    andXPar:"index"
  //                                                    andYPar:"close"],
  //                                              0.0,1/(2.0*5.29));
  // weekday (output Sun=0 to Sat=6) is linearly mapped low to high by:
  // weekday=linearLowToHigh(weekday,(double)SUNDAY,(double)SATURDAY)
  double tuesday = linearLowToHigh((double)TUESDAY,(double)SUNDAY,(double)SATURDAY);

  if((*squashedWeekday) < tuesday )
    {
      double adjustPercentage = (double)MONDAY_SIGNIFICANCE;

      [self adjust:inputA by:&adjustPercentage andOutput:output];
    }
  else
    *output = *inputA;
  return self;
}

- electionCycleWithInput:(double *)inputA andElectionYear:(double *)squashedElectionYear
 andOutput:(double *)output
{
  // for PSC.2: inputA=PSC.1
  // electionYear (output ...,88,89,...) is linearly mapped low to high by according to
election cycle:
  // electionYear=linearLowToHigh(year % 4,0,3); 0,3 are election years
  //
  // for PDC.3: inputA=PDC.2 electionYear
  double midPeriodBegin = linearLowToHigh(1,0,3);
  double midPeriodEnd = linearLowToHigh(2,0,3);
  double adjustPercentage = (double)ELECTION_SIGNIFICANCE;

  if( ((*squashedElectionYear) < midPeriodBegin) && ((*squashedElectionYear) > midPeriodE
nd) )
    [self adjust:inputA by:&adjustPercentage andOutput:output];
  else
    {
      adjustPercentage = - adjustPercentage;
      [self adjust:inputA by:&adjustPercentage andOutput:output];
    }
  return self;
}

- weightEquallyWithInputs:(double *)inputA :(double *)inputB andOutput:(double *)output
{
  const double weight2nd=0.5;

  (*output)=(1.0-weight2nd) * (*inputA) + weight2nd * (*inputB);
  return self;
}

- weightTwoThirdsInput:(double *)inputA withOneThirdInput:(double *)inputB andOutput:(dou
ble *)output
{
  const double weight2nd=1.0/3.0;

  (*output)=(1.0-weight2nd) * (*inputA) + weight2nd * (*inputB);
  return self;
}

- weightHighlySignificantInput:(double *)inputA withLessSignificantInput:(double *)inputB
 andOutput:(double *)output
{
  const double weight2nd=0.2;

  (*output)=(1.0-weight2nd) * (*inputA) + weight2nd * (*inputB);
  return self;
}

- weightEquallyAndNegateWithInputs:(double *)inputA :(double *)inputB andOutput:(double *
)output
{
  const double weight2nd=0.5;

  (*output)=(1.0-weight2nd) * (*inputA) + weight2nd * (*inputB);

  [self negate:output withReserved:(double *)NULL andOutput:output];
  return self;
}

- absoluteDifferenceOfInputs:(double *)inputA :(double *)inputB andOutput:(double *)outpu
t
{
  double diff = (*inputA) - (*inputB);

  (*output) = LOW + MAX( diff, -diff );
  return self;
}

@end
```

```
#import "Input.h"

typedef id (*RF_IMP)(id, SEL,const double*,const double*, double*);
typedef id (*JDF_IMP)(id, SEL,const double*,const double*,const double*,const double*, d
ouble*);
//typedef id (*MOD_IMP)(id, SEL, id, const double*, double*);

@interface Rule:Input
{
    id inputRulesIdentifiers; //matrix of Buttons
    id inputRules[2];
    id jointDistributionFunctionNameField;
    id ruleFunctionNameField;
}

- propagateChange:sender;
- batchMode:sender;
-(BOOL)isProcessedChangeAtInput: (unsigned int)inputNumber significant:sender;
-(id) write:(NXTypedStream *)stream;
-(id) read:(NXTypedStream *)stream;
//-(id)awake;

@end
```

```
#import "Rule.h"
#import "Statistics.h"
#import <appkit/Matrix.h>
#import <objc/List.h>
#import <stdlib.h>

@implementation Rule

- propagateChange:sender
{
    //do first what is necessary to update output if anything
    if([self isProcessedChangeAtInput:[inputRulesIdentifiers selectedRow] significant:se
nder])
        {
            //if there are significant changes, promote them
            [self outputUpdated];
        }
    return self;
}

- batchMode:sender
{
//    //do first what is necessary to update output if anything
//    if([self isProcessedChangeAtInput:[inputRulesIdentifiers selectedRow] significant:
sender])
//      {
//          //if there are significant changes, promote them
//          [self outputUpdated];
//      }
    // only propagate if not one of the input has already been propagated.
    if((inputRules[0]!=nil) && (inputRules[1]!=nil))
      [self batchModeUpdated];
    inputRules[[inputRulesIdentifiers selectedRow]] = nil;
    return self;
}

-(BOOL)isProcessedChangeAtInput: (unsigned int)inputNumber significant:sender
{
    SEL ruleFunctionSelector=sel_getUid((STR)[ruleFunctionNameField stringValue]);
    RF_IMP ruleFunction;
    SEL jointDistributionFunctionSelector=sel_getUid((STR)[jointDistributionFunctionName
Field stringValue]);
    JDF_IMP jointDistributionFunction;
    unsigned int inputRuleCount=[inputRulesIdentifiers cellCount];
    id *inputStatistics=NXZoneCalloc([self zone], inputRuleCount, sizeof(id));
    double *binSize=NXZoneCalloc([self zone], inputRuleCount, sizeof(double));
    double *counter=NXZoneCalloc([self zone], inputRuleCount, sizeof(double));
    double *upperBound=NXZoneCalloc([self zone], inputRuleCount, sizeof(double));
    double binSizeSelf=((double)(*[[self statistics] range]).size.width)/((double)[[self
 statistics] binCount]);
    unsigned int i;
    BOOL completelyInitializedSources=YES;

    //test if the selectors are valid
    if(!([self respondsTo:jointDistributionFunctionSelector]&&[self respondsTo:ruleFunct
ionSelector]))
        {
            char *tempString=NXZoneCalloc([self zone], 1023, sizeof(char));

            sprintf(tempString,"Either your Rule Function (RF) and/or \nJoint Distributi
on Density Function (JDF) \nis/are invalid in %s.\nCheck spelling!", [self title]);
            [self showError:tempString];
            return NO;
        }
    ruleFunction=(RF_IMP)[self methodFor:ruleFunctionSelector];
    jointDistributionFunction=(JDF_IMP)[self methodFor:jointDistributionFunctionSelector]
;
    //cache the sender in the inputRules, this is necessary, since we always get
    //only the pointer to the object that was updated, however not to the other
    //input, thus by caching, we have all pointers, when all inputs have changed
    //at least once (i.e have at least been initialized)
    inputRules[inputNumber]=sender;
    //calculate the neccesary binSizes, a binSize of 0.0 indicates that the
    //appropriate sender is unusable for use as input statistics provider
    for(i=0;i<inputRuleCount;i++)
        {
            if((inputRules[i]!=nil)&&([inputRules[i] respondsTo:@selector(statistics)]))
                {
                    inputStatistics[i]=[inputRules[i] statistics];
                    upperBound[i]=((double)(*([inputStatistics[i] range])).size.width)+((
double)(*([inputStatistics[i] range])).origin.x);
                    binSize[i]=((double)(*[inputStatistics[i] range]).size.width)/((doubl
e)[inputStatistics[i] binCount]);
                }
            else
                {
                    inputStatistics[i]=nil;
                    binSize[i]=0.0;
                    completelyInitializedSources=NO;
                }
        }
    if(completelyInitializedSources)
        {
            //do updates:
            //first save some time by preventing a constant update of the distribution
            //ATTENTION: calls to lockDistribution have to be balanced
            //           with calles to unlockDistribution similar
            //           to calls to lockFocus and unlockFocus...
            [[self statistics]lockDistribution];
            //then clear out the current output, since summing for integration has to
            //start out with a zero value
            [[self statistics]zeroOut];
            //now integrate:
            //should be done at some point in the future by means of a recursive
            //version of for() i.e should be a inputRuleCount deep nested loop
            //that calls at the innermost loop a function that takes inputRuleCount+1
            //arguments (i.e. all inputRuleCount inputs plus the output value) etc.
            //for now we use only two inputs and thus hardcode that...
            for(counter[0]=(((double)(*[inputStatistics[0] range]).origin.x)+(0.5*binSize
[0]));
                counter[0]<upperBound[0];
                counter[0]+=binSize[0])
                {
                    for(counter[1]=(((double)(*[inputStatistics[1] range]).origin.x)+(0.5
*binSize[1]));
                        counter[1]<upperBound[1];
                        counter[1]+=binSize[1])
                        {
                            double ruleFunctionResult, jointProbabilityDensity;

                            ruleFunction(self,
                                         ruleFunctionSelector,
                                         &counter[0],
                                         &counter[1],
                                         &ruleFunctionResult);
                            jointDistributionFunction(self,
                                                      jointDistributionFunctionSelector,
                                                      &counter[0],
```

```
                                                    [inputStatistics[0] densityAt:coun
ter[0]],
                                                    &counter[1],
                                                    [inputStatistics[1] densityAt:coun
ter[1]],
                                                    &jointProbabilityDensity);
                                [[self statistics]add:((jointProbabilityDensity*binSize[0]*b
inSize[1])/binSizeSelf) toDensityAt:ruleFunctionResult];
                                }
                    }
                //unlock the distribution and sync it with the new density function
                [[[self statistics]unlockDistribution]updateDistribution];
                //test for significance for speed up
                //can use variable in future implementations to have different
                //speed vs. precision trade-offs

                //return if changes were significant
                //YES for the time being
                return YES;
            }
        else
            return NO;
}

-(id) write:(NXTypedStream *)stream
{
    [super write:stream];
    NXWriteObjectReference(stream, inputRulesIdentifiers);
    NXWriteObjectReference(stream, jointDistributionFunctionNameField);
    NXWriteObjectReference(stream, ruleFunctionNameField);
    return self;
}

-(id) read:(NXTypedStream *)stream
{
    [super read:stream];
    inputRulesIdentifiers = NXReadObject(stream);
    jointDistributionFunctionNameField = NXReadObject(stream);
    ruleFunctionNameField = NXReadObject(stream);
    return self;
}

//-(id) awake
//{
//  [super awake];
//  return self;
//}

@end
```

```
#import <objc/Object.h>
#import <appkit/Box.h>
#import <appkit/graphics.h>

@interface StatisticsView:Box
{
    id bins;
    id binButtons;
    id displayModeButton;
    id inspectorButton;
    id valueForm;
    id statistics;
}

- (id)initFrame:(const NXRect *)frameRect;
- (id)initFrame:(const NXRect *)frameRect withStatistics:(id)someStatistics;
- (id)update:(id)someStatistics;
- showError: (const char *)errorMessage;
- displayValue:sender;
- changeDisplay:sender;
-(id)statistics;
-(id)freeStatistics;
-(id)write:(NXTypedStream *)stream;
-(id)read:(NXTypedStream *)stream;
//-(id)awake;

@end
```

```
#import "StatisticsView.h"
#import <objc/Storage.h>
#import <appkit/Application.h>
#import <appkit/Cell.h>
#import <appkit/Matrix.h>
#import <appkit/Button.h>
#import <appkit/Form.h>
#import <appkit/Panel.h>
#import <objc/List.h>
#import "Statistics.h"
#import "BinButtonCell.h"
#import "BinMatrix.h"

@implementation StatisticsView

- (id)initFrame:(const NXRect *)frameRect
{
    [super initFrame:frameRect];
    statistics=[[Statistics allocFromZone:[self zone]]init];
    bins=[statistics bins];
    return [self update:statistics];
}

- (id)initFrame:(const NXRect *)frameRect withStatistics:(id)someStatistics
{
    [self initFrame:frameRect];
    [self freeStatistics];
    statistics=someStatistics;
    return [self update:statistics];
}

/*
*- (id)update:(id)someStatistics
*{
*    NXRect originalFrameSize;
*    unsigned int i;
*
*    statistics=someStatistics;
*    bins=[statistics bins];
*    //due to strange behaviour of the NeXTstep system, I can't use nested message passi
ng and one single NXRect...
*    // send bug report as soon as possible
*    [[binButtons getBounds:&originalFrameSize]display];
*    [[binButtons renewRows:1 cols:[bins count]]display];
*    [[binButtons sizeToCells]display];
*    [[binButtons sizeTo:originalFrameSize.size.width:originalFrameSize.size.height]disp
lay];
*    for(i=0;i<[bins count];i++)
*       {
*           [[binButtons putCell:
*               [[[[BinButtonCell allocFromZone:[self zone]]init]
*                  setBinValue:*((double *)[bins elementAt:i])]
*                  setMaxGraphValue:(*[statistics range]).size.height]
*               at:0:i]
*           free];
*           [[binButtons setAction:@selector(displayValue:)at:0:i]setTarget:self at:0:i]
;
*       }
* //    sprintf(windowTitle,"Connections from %i to %i",tag,(tag+1));
* //    [self setTitle:windowTitle];
*    NXPing();
*    return self;
*}
*/

- (id)update:(id)someStatistics
{
    NXRect originalFrameSize;
    unsigned int i;

    statistics=someStatistics;
    bins=[statistics bins];
    //due to strange behaviour of the NeXTstep system, I can't use nested message passing
 and one single NXRect...
    // send bug report as soon as possible
    [binButtons getBounds:&originalFrameSize];
    [binButtons renewRows:1 cols:[bins count]];
    [binButtons sizeToCells];
    [binButtons sizeTo:originalFrameSize.size.width:originalFrameSize.size.height];
    for(i=0;i<[bins count];i++)
        {
            [[binButtons putCell:
                [[[[BinButtonCell allocFromZone:[self zone]]init]
                   setBinValue:*((double *)[bins elementAt:i])]
                   setMaxGraphValue:(*[statistics range]).size.height]
                at:0:i]
            free];
            [[binButtons setAction:@selector(displayValue:)at:0:i]setTarget:self at:0:i];
        }
//    sprintf(windowTitle,"Connections from %i to %i",tag,(tag+1));
//    [self setTitle:windowTitle];
    return [self update];
}

- showError: (const char *)errorMessage
{
  NXRunAlertPanel(NULL, errorMessage,"OK", NULL, NULL);
  // TODO: add string table support
  //  NXRunAlertPanel(NULL, errorMessage,
  //              [stringSet valueForStringKey:"OK"], NULL, NULL);
  return self;
}

- displayValue:sender
{
    [valueForm setDoubleValue:*((double *)[bins elementAt:[sender selectedCol]])];
    return self;
}

- changeDisplay:sender
{
    //for pop-up lists:
    //[statistics setDensityActive:[[sender selectedCell]tag]];
    //for toggle buttons:
    [statistics setDensityActive:[sender intValue]];
    return [self update:statistics];
}

-(id)statistics
{
    return statistics;
}

-(id) freeStatistics
{
    if(statistics != nil)
        {
            [statistics free];
```

```
            statistics = nil;
        );
    return self;
}

-(id) write:(NXTypedStream *)stream
{
    [super write:stream];
    NXWriteObject(stream, statistics);
    NXWriteObjectReference(stream, bins);    // not necessary, see awake
    NXWriteObjectReference(stream, binButtons);
    NXWriteObjectReference(stream, displayModeButton);
    NXWriteObjectReference(stream, inspectorButton);
    NXWriteObjectReference(stream, valueForm);
    return self;
}

-(id) read:(NXTypedStream *)stream
{
    [super read:stream];
    statistics=NXReadObject(stream);
    bins=NXReadObject(stream);              // not necessary, see awake
    binButtons=NXReadObject(stream);
    displayModeButton=NXReadObject(stream);
    inspectorButton=NXReadObject(stream);
    valueForm=NXReadObject(stream);
    return self;
}

//-(id)awake
//{
//    [super awake];
//    bins=[statistics bins];
//    return [self update:statistics];
//}

@end
```

```
#import <objc/Object.h>
#import <appkit/graphics.h>

//todo:
// add support for updating the distribution slots

@interface Statistics:Object
{
    id densityBins;             //Storage object
    id distributionBins;        //Storage object
    NXRect   densityRange;
    NXRect   distributionRange; // the origin.x and size.width part
                                // always should be the same as for the
                                // densityRange, also for all purposes,
                                // origin.y should be 0.0
    double   binSize;
    BOOL densityActive;
    BOOL distributionLocked;
}

-(id) init;
-(id) initWithBins:(unsigned int)count;
-(id) initNormalWithMean:(double)aMean
      stdDev:(double)aStdDeviation
      andBins:(unsigned int)count;
-(id) fillWithMean:(double)aMean
      stdDev:(double)aStdDeviation;
-(id) fillWithBiModality:(double)aBiModality
      stdDev:(double)aStdDeviation;
-(id) zeroOut;
-(id) lockDistribution;
-(id) unlockDistribution;
-(id) distributionBins;
-(id) densityBins;
-(id) bins;
-(id) setBinCount:(unsigned int)count;
-(unsigned int) binCount;
-(NXRect *) densityRange;
-(NXRect *) distributionRange;
-(NXRect *) range;
-(id) setDensityActive:(BOOL)aBool;
-(BOOL) densityActive;
-(id) setDensityAt:(double)x to:(double)y;
-(id) setDistributionAt:(double)x to:(double)y;
-(id) add:(double)deltaY toDensityAt:(double)x;
-(double *) densityAt:(double)x;
-(double *) distributionAt:(double)x;
-(id) updateDistribution;
-(double)probabilityAt:(double)x;
-(double)mean;
-(double)variance;
-(double)standardDeviation;
-(id) free;
-(id) write:(NXTypedStream *)stream;
-(id) read:(NXTypedStream *)stream;
-(id) awake;

@end
```

```
#import "Statistics.h"
#import <math.h>
#import <objc/Storage.h>
#import "StatisticsView.h"
#import <float.h>

#define DEFAULT_BINS 50
#define X_RANGE_ORIGIN -1.5
#define X_RANGE_SIZE 3.0
#define Y_RANGE_SIZE_DENSITY 3.0

@implementation Statistics

double normalDistributionDensity(double x,double aMean,double aStdDeviation)
{
    return (1.0/(exp(pow(-aMean+x,2)/(2*pow(aStdDeviation,2)))*sqrt(2*M_PI)*aStdDeviatio
n));
}

double biModalDistributionDensity(double x,double aBiModality,double aStdDeviation)
{
    return ((1.0/(exp(pow(-aBiModality+x,2)/(2*pow(aStdDeviation,2)))*sqrt(2*M_PI)*aStdD
eviation)) +
            (1.0/(exp(pow(aBiModality+x,2)/(2*pow(aStdDeviation,2)))*sqrt(2*M_PI)*aStdDev
iation)) ) / 2;
}

-(id) init
{
    return [self initNormalWithMean:0.0 stdDev:.5 andBins:DEFAULT_BINS];
}

-(id) initWithBins:(unsigned int)count
{
    [super init];
    densityBins=[[Storage allocFromZone:[self zone]]
                            initCount:count
                            elementSize:sizeof(double)
                            description:"d"];
    distributionBins=[[Storage allocFromZone:[self zone]]
                                initCount:count
                                elementSize:sizeof(double)
                                description:"d"];
    densityActive=YES;
    densityRange.origin.x=(X_RANGE_ORIGIN);
    densityRange.origin.y=0.0;
    densityRange.size.width=(X_RANGE_SIZE);
    densityRange.size.height=(Y_RANGE_SIZE_DENSITY);
    distributionRange=densityRange;
    distributionRange.size.height=1.0;
    binSize=((double)densityRange.size.width)/(double)count;
    return self;

}

-(id)initNormalWithMean:(double)aMean stdDev:(double)aStdDeviation andBins:(unsigned int
)count
{
    return [[self initWithBins:count]fillWithMean:aMean stdDev:aStdDeviation];
}

-(id)fillWithMean:(double)aMean stdDev:(double)aStdDeviation
{
    unsigned int i, binCount=[densityBins count];
// removed by agm:    double *binValue;

    for(i=0;i<binCount;i++)
        {
            double x=(double)((double)densityRange.origin.x+((((double)densityRange.size.
width) / ((double)[densityBins count]))* (((double)i) + (double)0.5)));
// removed by agm:          binValue=[densityBins elementAt:i];

            (*(double *)[densityBins elementAt:i])=normalDistributionDensity(x,aMean,aStd
Deviation);
        };
    //here init the distribution....
    return [self updateDistribution];
}

-(id) fillWithBiModality:(double)aBiModality
      stdDev:(double)aStdDeviation
{
    unsigned int i, binCount=[densityBins count];

    for(i=0;i<binCount;i++)
        {
            double x=(double)((double)densityRange.origin.x+((((double)densityRange.size.
width) / ((double)[densityBins count]))* (((double)i) + (double)0.5)));

            (*(double *)[densityBins elementAt:i])=biModalDistributionDensity(x,aBiModali
ty,aStdDeviation);
        };
    //here init the distribution....
    return [self updateDistribution];

}

-(id) zeroOut
{
    unsigned int count=[self binCount];
    unsigned int i;

    for(i=0;i<count;i++)
        *((double *)[densityBins elementAt:i])=(double)0.0;
    return [self updateDistribution];
}

-(id) lockDistribution
{
    distributionLocked=YES;
    return self;
}

-(id) unlockDistribution
{
    distributionLocked=NO;
    return self;
}

-(id) distributionBins
{
    return distributionBins;
}

-(id) densityBins
{
    return densityBins;
```

```
-(id) bins
{
    if(densityActive)
        return densityBins;
    else
        return distributionBins;
}

-(id) setBinCount:(unsigned int)count
{
    unsigned int i;
    double newBinSize=(double)densityRange.size.width/(double)count;

    id newDensityBins=[[Storage allocFromZone:[self zone]]
                               initCount:count
                               elementSize:sizeof(double)
                               description:"d"];
    id newDistributionBins=[[Storage allocFromZone:[self zone]]
                               initCount:count
                               elementSize:sizeof(double)
                               description:"d"];

    for(i=0;i<count;i++)
        {
            *((double *)[newDensityBins elementAt:i])=
                        *([self densityAt:(((double)(densityRange.origin.x))
                                                        +(((double)i)*newBinSize)
                                                        +(0.5*newBinSize))]);
            (*(double *)[newDistributionBins elementAt:i])=
                        (*[self distributionAt:(((double)(distributionRange.origin.x))
                                                                +(((double)i)*n
ewBinSize)
                                                                +(0.5*newBinSiz
e))]);
        }
    [densityBins free];
    densityBins=newDensityBins;
    [distributionBins free];
    distributionBins=newDistributionBins;
    binSize=newBinSize;
    return self;
}

-(unsigned int)binCount
{
    return [[self bins]count];
}

-(NXRect *)densityRange
{
    return &densityRange;
}

-(NXRect *)distributionRange
{
    return &distributionRange;
}

-(NXRect *)range
{
    if(densityActive)
        return &densityRange;
    else
        return &distributionRange;
}

-(id) setDensityActive:(BOOL)aBool
{
    densityActive=aBool;
    return self;
}

-(BOOL)densityActive
{
    return densityActive;
}

-(id) setDensityAt:(double)x to:(double)y
{
    int bin=floor((x-(double)densityRange.origin.x)/binSize);

    if((bin>=0)&&(bin<[self binCount]))
        {
            *((double *)[densityBins elementAt:(unsigned int)bin])=y;
        }
    return self;
}

-(id) setDistributionAt:(double)x to:(double)y
{
    int bin=floor((x-(double)distributionRange.origin.x)/binSize);

    if((bin>=0)&&(bin<[self binCount]))
        {
            *((double *)[distributionBins elementAt:(unsigned int)bin])=y;
        }
    return self;
}

-(id) add:(double)deltaY toDensityAt:(double)x
{
    int bin=floor((x-(double)densityRange.origin.x)/binSize);

    if((bin>=0)&&(bin<[self binCount]))
        {
            *((double *)[densityBins elementAt:(unsigned int)bin])+=deltaY;
        }
    return [self updateDistribution];
}

-(double *) densityAt:(double)x
{
    int bin=floor((x-(double)densityRange.origin.x)/binSize);

    if((bin>=0)&&(bin<[self binCount]))
        return [densityBins elementAt:(unsigned int)bin];
    else
        return NULL;
}

-(double *) distributionAt:(double)x
{
    int bin=floor((x-(double)distributionRange.origin.x)/binSize);

    if((bin>=0)&&(bin<[self binCount]))
        return [distributionBins elementAt:(unsigned int)bin];
    else
```

```
        return NULL;
}

-(id) updateDistribution
{
    if(distributionLocked)
        return self;
    else
        {
            unsigned int i;

            (*(double *)[distributionBins elementAt:0])=(*(double *)[densityBins element
At:0])*binSize;
            for(i=1;i<[distributionBins count];i++)
                {
                    (*(double *)[distributionBins elementAt:i])=((*(double *)[densityBin
s elementAt:i])*binSize)+(*(double *)[distributionBins elementAt:(i-1)]);
                };
            return self;
        }
}

-(double)probabilityAt:(double)x
{
  double *distAtX=(double *)([self distributionAt:x]);
  double *distBeforeX=(double *)([self distributionAt:(x - binSize)]);

 if(distBeforeX==NULL)
   return (*distAtX);
 else
   return (*distAtX) - (*distBeforeX);
}

-(double)mean
{
//  mean=sum( p(x) * x )

  double mean=0.0;
  double probSum=0.0;
  unsigned int i, binCount=[densityBins count];

  double x=distributionRange.origin.x + (0.5 * binSize);

  for(i=0;i<binCount;i++)
    {
      double probOfX=[self probabilityAt:x];

      mean+= probOfX * x;
      probSum+=probOfX;
      x+=binSize;
    }
  // ideally, probSum should be 1.0, but in case it's not, adjust it.
  mean = mean / probSum;

  return mean;
}

-(double)variance
{
// sum( ( x - mean )^2 )
  double mean=[self mean];
  double varSum=0.0, probSum=0.0;
  unsigned int i, binCount=[densityBins count];

  double x=distributionRange.origin.x + (0.5 * binSize);

  for(i=0;i<binCount;i++)
    {
      double probOfX=[self probabilityAt:x];

      varSum+= probOfX * ((x - mean) * (x - mean));
      probSum+=probOfX;
      x+=binSize;
    }
  // ideally, probSum should be 1.0, but in case it's not, adjust it.
// printf("probSum variance: %f\n",probSum);
  varSum = varSum / probSum;

  return varSum;
}

-(double)standardDeviation
{
  return sqrt([self variance]);
}

-(id) free
{
    [densityBins free];
    [distributionBins free];
    return [super free];
}

-(id) write:(NXTypedStream *)stream
{
    [super write:stream];
    NXWriteObject(stream, densityBins);
    NXWriteObject(stream, distributionBins);
    NXWriteRect(stream, &densityRange);
    NXWriteRect(stream, &distributionRange);
    NXWriteTypes(stream,"dc", &binSize, &densityActive);
    return self;
}

-(id) read:(NXTypedStream *)stream
{
    [super read:stream];
    densityBins=NXReadObject(stream);
    distributionBins=NXReadObject(stream);
    NXReadRect(stream, &densityRange);
    NXReadRect(stream, &distributionRange);
    NXReadTypes(stream,"dc", &binSize, &densityActive);
    return self;
}

-(id) awake
{
    [super awake];
    distributionLocked=NO;
    return self;
}

@end
```