SmileX

An ActiveX Decision-Analytic Reasoning Engine and Its Application to Evaluation of Credit Applicants

C.P.R.Thijssen

1999

Delft University of Technology
Faculty of InformationTechnology and Systems
Knowledge Based Systems Group

University of Pittsburgh
School of Information Sciences
Decision Systems Laboratory

SmileX

An ActiveX Decision-Analytic Reasoning Engine and Its Application to Evaluation of Credit Applicants

Abstract

In the first part of this project an ActiveX control was developed that wraps an inference engine. This control, called SmileX, can be inserted in web-pages, applications made in several different languages and other software supporting ActiveX techniques. It implements an easy-access interface to the algorithms and procedures available in the C++ library SMILE.

In the second part of this project SmileX was applied to the decision problem of evaluating creditworthiness in consumer credit. A Bayesian network model was built in order to support decisions in this field. A consumer-oriented web-interface and a bank-oriented Excel-interface were developed to illustrate the use of both the model and the SmileX control developed in the first part of this project.

Graduation committee:
Prof. Dr. H. Koppelaar
Dr. Drs. L.J.M Rothkrantz (supervisor)
Prof. Dr. Ir. E.J.H. Kerckhoffs

Printed in Delft, The Netherlands

For my parents

Preface

Aims and audience

This report describes the Masters thesis I did to complete my studies in Computer Science at the Delft University of Tecnology. It is intended as a final report for this project and gives the reader insight in the work I did during the last 10 months.

The report can be read by anyone who is interested in applications of artificial intelligence techniques. Some readers may want to skip the first part that delves into the details of implementing an engine for performing evaluation of Bayesian networks and influence diagrams. These readers should skip the chapters 2 and 3.

Perople interested in Bayesian networks, influence diagrams and applying artificial intelleigence techniques to the problem of evaluating creditworthiness in consumer credit should read the second part consisting of the chapters 4 to 7.

The last chapter contains conclusions and recommendations of the project. People interested only in the outcome of the project can try to read this chapter first.

Acknowledgements

This project could not have been completed without the help and advise of two people. Prof. M.J. Drużdżel has offered me the opportunity to be a part of his laboratory. I have enjoyed being part of his insipring research group and would like to thank him for this and everything else he has done for me. However far away Drs. dr. L.J.M. Rothkrantz has been the one that kept me on track. Especially in the last weeks he has given me the motivation to go ahead and complete this project. He has also initiated everything by putting me in touch with Prof. M.J. Drużdżel and helped me with the preparations for my stay in Pittsburgh.

Also my family has played a big role in this project. My father and mother have always supported me in every way and have shown a lot of patience. They have been waiting quite a while for this report, but never stopped believing in me. My brother Mark-Johan and sister Marie-Louise are to be thanked as well. They have been good role-models for their little brother and have also supported me in every possible way.

Last but not least my friends and colleagues in Delft and Pittsburgh deserve to be mentioned in this section. After all they have been putting up with me for quite a while! There's a few people that I'd like to mention by name: Agnieszka, Bertil, Dani, Denver, John, Nicole, Rutger, Sidney and Tessa. Thanks everybody!

C.P.R.Thijssen, October 10, 1999

For reasons of simplicity, the pronouns 'he', 'him', 'his' etc. are used to relate to both male and female throughout the report

Contents

List of Figures

Chapter 1

Introduction

1.1 About the report and the project

This report is the result of a Master's Thesis project for the Computer Science studies at the Delft University of Technology. The project was conducted at the School of Information Sciences of the University of Pittsburgh and is the result of an agreement of common research and a student exchange program between these two universities. In this section we introduce the organisational structure of the environment in which this project was done.

1.1.1 Decision Systems Laboratory

The Decision Systems Laboratory is a research group of the School of Information Sciences of the University of Pittsburgh. The group is headed by Prof. M.J. Drużdżel, assistant professor at the School of Information Sciences, Intelligent Systems Program, and Medical Informatics Training Program, University of Pittsburgh. Besides his work at the University of Pittsburgh he teaches courses on Decision Analysis and Decision Support systems at Carnegie Mellon University.

The DSL focuses on the use of techniques and systems that support decision making under uncertainty. Probabilistic, decision- theoretic and econometric techniques are combined with artificial intelligence approaches in order to research and develop decision support systems. The DSL offered the perfect environment for the research and development done in this project.

1.1.2 Knowledge Based Systems

The group Knowledge Based Systems at the Faculty of Information Technology and Systems is part of the Technical Informatics Department of the Faculty of Information Technology and Systems. The group is headed by Prof. Dr. H. Koppelaar and researches the different fields of Artificial Intelligence.

This project was done under the supervision of Drs. dr. L.J.M. Rothkrantz , associate professor of the Knowledge Based Systems group. He is also the project leader of the research project Multimodal Interfaces. In this project one uses traditional AI techniques like expert systems and connectionist techniques like neural networks in order to analyse, interpret and process (sensor-) data. This data is obtained from human interaction with multimedia devices.

1.2 Organisation of the report

The project consists of two parts: creating an AI software module (SmileX) and applying this module to the decision problem of evaluating consumer credit. This report has as its first two sections a description of these two pieces. The third part contains a conclusion and recommendations for further research and development.

SmileX

The first part of the project was to create an easy to use application programmers interface for the complex algorithms used to calculate decision support information from probabilistic models. This process is described in detail in the first part of this report.

Credit evaluation

The second part of the project involved creating a model that helps banks in evaluating loan applications. In order to test the software module developed in the first part, and to establish an opinion about its useability, this credit-evaluation model was wrapped in a full-featured application. Part II of this report describes this.

Conclusion and recommendations

This final chapter evaluates the project and puts it in perspective. It makes some concluding remarks and gives ways in which way this research and development project can be extended or improved.

1.3 Decision Support Systems

1.3.1 Overview

With the current status of computers and techniques one can have automated systems that help in making decisions. These systems (usually computer-programs) are called decision support systems. This is a general term for a vast field of techniques.

Examples of decision support systems

Neural networks Neural networks are trained by feeding it with correct case-decision pairs. After training it a case is posed to the system, which is evaluated using connectionist algorithms modeled after processes in the brain.

Rule-based systems Rule-based systems are based on evaluating logical rules. Given a set of rules and a case to evaluate, the system can help decide on the optimal alternative.

Normative systems In this field one uses normative probabilistic and statistical techniques to evaluate cases. By building a model of reality, including all factors that have influence on the decision one can make a system that calculates the optimal decision.

1.3.2 Bayesian Networks and Influence Diagrams

In this project we make use of these normative systems. In order to support a decision, this decision has to be modeled first. This can be done by using a modeling technique, among which Bayesian networks and influence diagrams are the most used.

Bayesian networks

Bayesian networks are directed acyclic graphs in which nodes represent variables that have influence on the decision to be taken. These influences are called causal relations. Arcs between these nodes represent the probabilistic dependencies between the variables.

A decision problem can be modeled by a Bayesian network by entering as variables the factors that play a role in the decision process. The arcs in the graph represent the causal dependencies. Bayesian networks are also called *(Bayesian) belief networks* or *causal (probabilistic) networks*

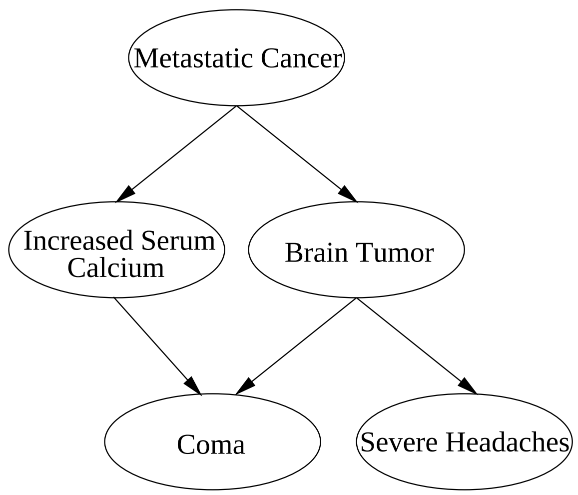


Figure 1.1: An example Bayesian network

Brain Tumor / Metastatic Cancer	**Present**	**Absent**
Present	0.2	0.05
Absent	0.8	0.95

Table 1.1: Definition of node *Brain Tumor*

This network [8] models a (simplified) medical diagnostic process. With this model the chance that a patient will go into coma can be predicted, based on information about his health (having a brain-tumor, having an increased serum calcium etc.). When we look at the arc from the node Metastatic Cancer to Brain Tumor we can draw the conclusion that having a Metastatic Cancer has impact on the chance of having a brain tumor. In order to make probabilistic predictions about going into a coma we have to define the different probabilities.

For every node such a table must be defined in order to make correct predictions. The numbers that have to be put into these table have to be obtained either from test-data or from an expert in the specific field. When all these definition values have been entered, one can start obtaining results from the model. For example, if we know that a certain patient has Metastatic Cancer and a brain tumor, we enter this information in the model and it will calculate the exact chance of the patient going into a coma. These calculations are done by applying the following Bayesian theorem to the network-variables.

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

also written as:

$$P(A|B) = P(B|A) * \frac{P(A)}{P(B)}$$

The process of applying this fundamental theorem on all the variables in the network is called *performing inference*.

Influence diagrams

Influence diagrams are a superset of Bayesian networks. Where Bayesian networks can only model variables in a decision environment, influence diagrams can model the whole decision process. In addition to the variables there are entities that represent the decision itself, values for defining the appreciation of different alternatives etc.

This influence diagram [1] represents the decision process of a pinball-machine collector considering the options to turn his hobby into a business or selling the collection. The rectangular shapes in this model represent decisions. The arcs between the decisions are not causal relationships, but rather temporal. For example, before being able to decide on the location where the machines can be placed (Bar's or Arcade's) one has to decide first on whether one wants to start the business at all.

[1] This influence diagram has been developed by Michael T. Filipiak, Charles Neville and Joseph C. Wynn as a Decision Analysis class project at Carnegie Mellon Univeristy

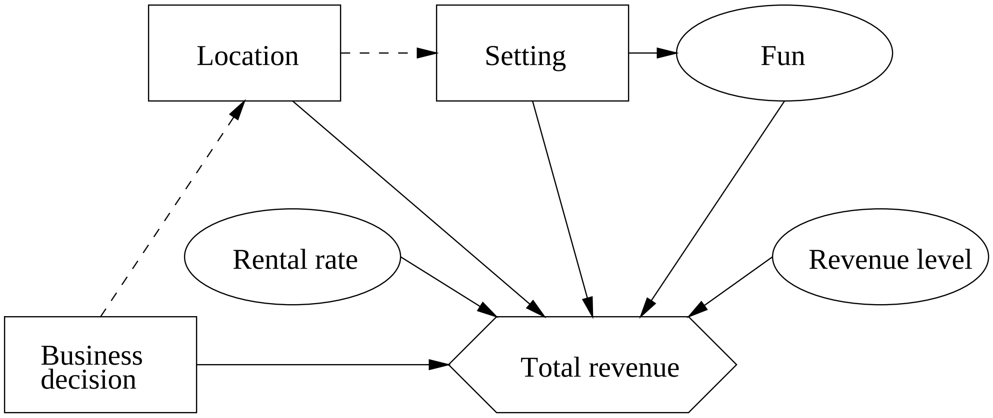


Figure 1.2: An example influence diagram

The diamond shaped node *Total revenue* represents the value of the decision process. In this case the value is a number that represents the dollar value of the different options. In other cases it can represent the level of satisfaction or another scale of measurement upon which the alternatives can be evaluated and compared. In decision theory this type of nodes is also called utility nodes.

In general influence diagrams are usefull when the whole decision process has to be modeled (in this case because there is more than one decision to take). Other cases can be modeled by a Bayesian network model. In these cases the model can easily be adapted to model the whole decision process. For instance, in the case where a bank has to decide on whether or not to extend a customer credit, there is no need to model the whole decision process, there is only one final decision. By extending the model with a decision and a value node it can represent the whole decision 'process'.

1.4 SMILE and GeNIe

This section describes the software modules, developed by the DSL, in order to give the reader of this report more insight in which environment this project was done.

1.4.1 SMILE

The core module of all software developed in the DSL is SMILE, which is an acronym for Structural Modeling, Inference, and Learning Engine. As the name suggests, it implements a set of functions that can help you build Bayesian networks and influence diagrams, and, of course, generate decision support information from these models. This engine is an Application Programmers Interface (API) for programmers that want to make use of probabilistic reasoning in their programs.

With this engine one can create proprietary interfaces for different kinds of models.

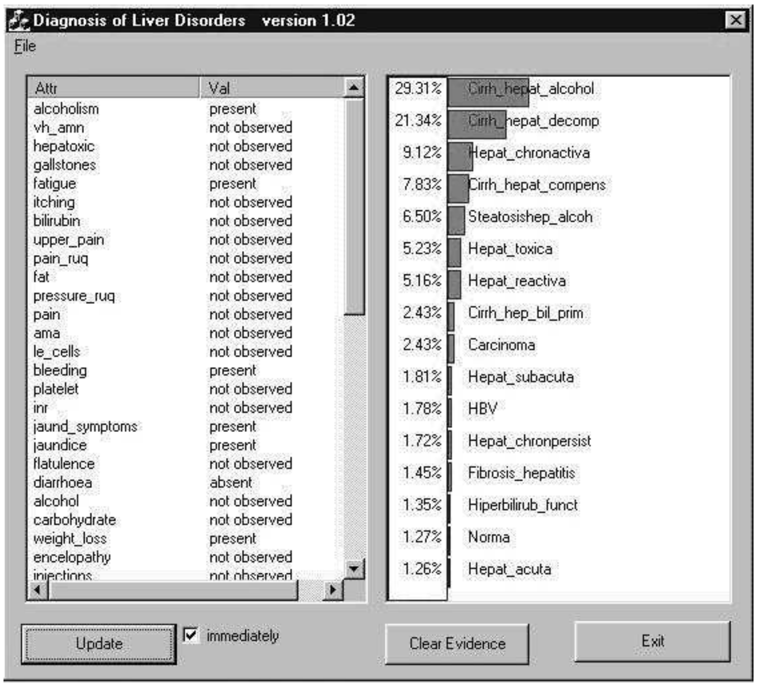


Figure 1.3: A C++ program using the SMILE C++ library

In the first part of this project, this engine is encapsulated in some sort of wrapper, in order to make it easier accessible and available for more programmers.

1.4.2 GeNIe

GeNIe is an end-user interface to this engine and primarily intended to be a development platform for probabilistic models. GeNIeis short for Graphical Network Interface and , as would be expected from the name, gives the user the possibility to create, modify and make calculations with these models, through a so-called Graphical User Interface (GUI).

GeNIe has over 1400 users worldwide as of August 1999. It is also used as an educational tool at universities, to support students in classes about decision analysis and decision support systems.

The wrapper discussed above does not give programmers the ability to create and modify Bayesian networks or influence diagrams. It rather requires the programmer to develop the model in GeNIe, after which it can be used (unmodified) in decision support applications.

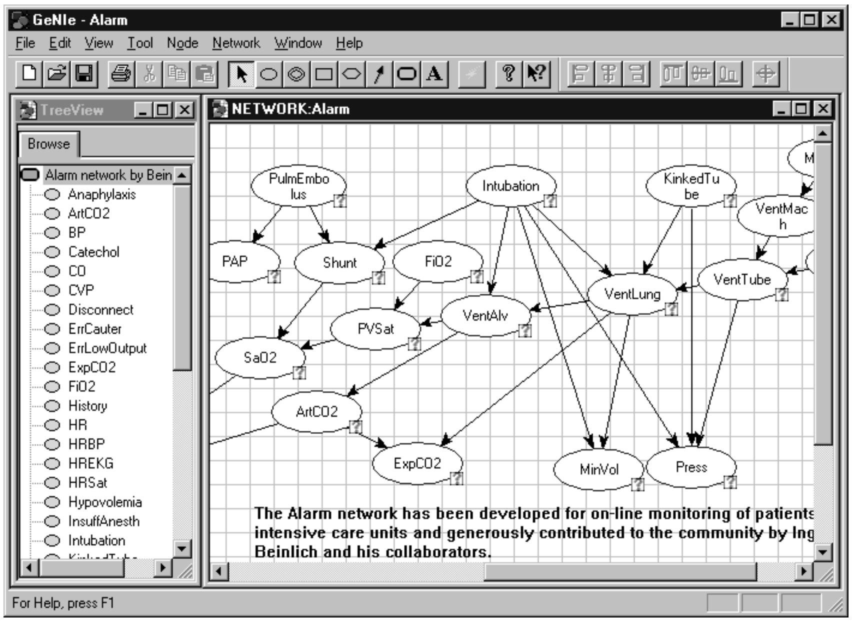


Figure 1.4: Schematic overview of GeNIe

1.4.3 Properties of GeNIe and SMILE

In order to be able to talk about certain topics in the rest of this report, the reader should be familiar with some jargon. GeNIe and SMILE use some definitions and terms that will be explained in the following.

Model A model in GeNIe and SMILE is the representation of a Bayesian network or an influence diagram. In GeNIe it is possible to graphically design a model by drawing a network with the available drawing tools. Typically a model consists of nodes and arcs.

Node A node represents a variable in a model. There are several types of nodes that are all displayed different (oval, diamond shape, box etc.)

Arcs An arc is used to show that one node influences or is influenced by another node. Dotted arcs however are used to determine the temporal behaviour of nodes, i.e. which variable is determined before another.

Node definition The node definition defines the 'behaviour' of the node. The definition contains the information that describes the way that the variable represented by the node interacts and/or is influenced by other nodes. Usually the definition of a node consists of the different chances of the states that the variable can assume.

Node value The value of a node is the information that is generated by performing inference on the model.

Submodel A sub-model is a set of nodes that intuitively form a group. Usually a submodel can be seen as some kind of *black box* or a single node that (internally) consists of a network.

Evidence Evidence is the information about nodes that tells the network that there is no doubt about the resulting state of the node. The inference algorithms can then take these facts into account and compute a more precise result. For instance: A node 'gender' can either be *male* or *female*. In the definition of the node it says that there is a 49% chance of a person being a male and a 51% chance of a person being a female. When we know for a fact that a person is a female, we can more or less 'disable' the chances by setting the evidence for this node to *female*.

Targets A node can either be a target node or not. A node that is a target node will contain the correct node value after performing inference on the model. By only setting the nodes as a target that you need to base your decision on, you can shorten the time the inference algorithm needs to calculate the node values. Nodes that are not a target may or may not contain correct node values after the inference is completed, depending on the need to calculate these values in order to determine the node values of the nodes that *are* target nodes.

1.4.4 Restrictions of GeNIe and SMILE

GeNIe and SMILE also have certain restrictions, that have great impact on this project.

No support for continuous variables This means that all variables in the models, that are developed with SMILE and GeNIe must consist only of discrete variables. Cases in which continuous variables are required have to be adjusted to contain discretized continuous variables.

For example, the age of a person can not be defined in the model as an integer. One has to define groups of ages (e.g. 0–18, 19–65, 66–Higher) and in this way discretize the variable age. Instead of entering the exact age of a person, one has to tell the model to which group the person belongs.

Evidence can not be saved It is impossible to save the evidence in the models in GeNIe and SMILE. This means that, once the model is defined there is no way to save the 'preset' option of a node.

For example, when we require a node (*gender*) to have evidence (*male* or *female*), there is no way of defining a default value for this node, only after the model is loaded we can manually set the evidence for a node.

Part I

SmileX, an ActiveX version of SMILE

Chapter 2

Creating an easy-access inference engine

This chapter describes the process of deciding on the best way to create an easy-access software module that provides programmers with the ability to use probabilistic reasoning in their programs.

2.1 Project goals

This section describes the goals of this project. In short there are three subgoals, all related to problems that have arised by using the existing software.

1. At the onset of my project, it was only possible to make use of the implemented algorithms and techniques by making a program in C++. One can use the functions and procedures in the C++-library by linking the library to the program. It would be a big advantage if not only C++ programmers could make use of the techniques offered by SMILE but also other programmers. In this context we mention some of the most popular programming languages: Java, Visual Basic, Pascal, C(++) etc.

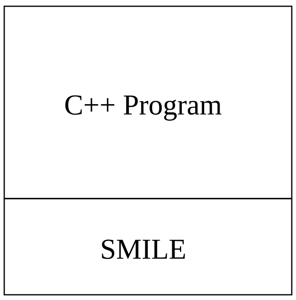


Figure 2.1: A C++ program using the SMILE C++ library

2. The second point of interest is the big expansion of web-applications. These days it is more and more necesary to make your applications *web-enabled* in order to keep up with the rapid development of the Internet. Some web-applications will also want to make use of the Bayesian network and influence diagram algorithms implemented in SMILE. Making this possible will enable web-site developers to use results of these algorithms and techniques in their web-sites. A good example of a web-application using Bayesian network techniques is a search engine that processes queries in natural language.

3. The third and very important aspect of this project is to develop a simple and easy to understand set of procedures and functions in order to offer developers a quick way to use SMILE without spending lots of time understanding the internal details of Bayesian networks and inference algorithms. This set of procedures and functions is often called the *application programmers interface* (API) and can be seen as the way that programmers 'talk' to SMILE.

2.2 The rewriting option

Starting from scratch by rewriting the whole engine could be a way to reach the goals of this project. Since rewriting the whole engine would require a lot more time than alotted for this project it was not considered to be a realistic option. This section summarizes the different options of rewriting shortly.

2.2.1 Java

One of the great advantages of Java is the 'write once, run anywhere' property of the language. This means that a Java program written on one architecture can also run (and generate the same results) on another processor and/or operating system.

Rewriting the engine in Java would be a good opportunity to make the Bayesian network algorithms available in many operating environments. One of the main advantages of an engine written in Java is that this engine can be used in the same form on any platform. An engine written in C++ has to be recompiled for every different platform in order to be useable in a C++ program on that platform. At this time the original C++ library is offered for three different platforms. It is compiled for a DOS/Windows environment, the Solaris operating system by SUN Microsystems and finally a Linux version.

Any Java program, regardless of the architecture it runs on, could make use of the Java-based Bayesian network engine. This is a good point considering the fact that one of the goals of the project is that the algorithms have to be used in many computing environments.

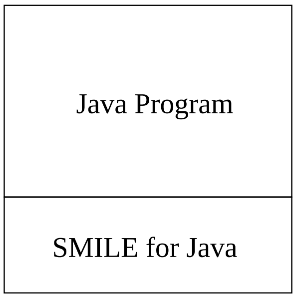


Figure 2.2: A Java program using the SMILE library ported to Java

With a Java engine it would also be easy to use the algorithms in a World-Wide-Web environment. It is possible to incorporate a special kind of Java programs (so-calles applets) in a web-page. The Java-engine can be incorporated into one of these applets and thus enable web-designers to make use of the therein implemented algorithms. The web-requirement would thereby be met.

The programmer of the Java version of SMILE can make the API of that library (or class as it is called in Java) as easy or hard-to-use as he wants. By not implementing all the functionality available in the C++ library it is easy to simplify the use of the Java-version. It should be noted however that the Bayesian network algorithms require a large chunck of the available functions present in the library. All the functions needed by these algorithms will have to be translated into Java. This is exactly why rewriting in Java would be very time-consuming.

Another issue that is worth mentinoing is that the algorithms in SMILE are computationally very intensive. Since Java is quite slower than C++ it is most likely that SMILE will never be ported to Java.

2.2.2 Component Object Model

The Component Object Model (COM) can be seen as a programming paradigm. By implementing software modules, like the SMILE-engine, according to the rules defined in the COM specifications, it becomes possible to use the engine from a vast range of programming environments under the Microsoft Windows operating system. For more details on COM and a list of these environments see 3.1.1.

COM provides a way to detach the language in which the engine is programmed from the language in which the program is written that uses this engine. When a (COM) rewrite is considered all the functions and elements in the engine would have to be rewritten with this set of rules in mind.

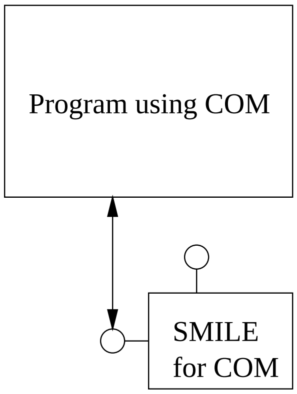


Figure 2.3: A COM-based program interacting with the COM version of the SMILE library

Like in Java, the API is independent of rewriting or not. However, later in this chapter we will see that an ActiveX control is a special sort of COM-object. One of the reasons we can use this ActiveX control, is that we want a limited API, that does not implement all the available functions.

With the use of COM it also becomes possible to use the functions in the engine over a network (and thus the Internet), which is also one of the major requirements in this project. Actually this technique is called Distributed COM (DCOM) and enables the programmer to call a function on another computer.

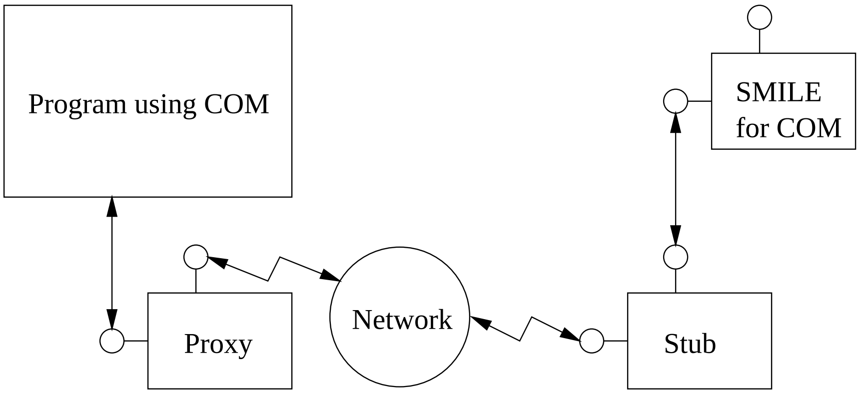


Figure 2.4: COM technology over a network (e.g. the Internet)

2.3 Encapsulating the engine

Rewriting the entire engine in order to accomplish the goals is quite a lot of work. By encapsulating the engine in some kind of layer we may be able to get the best of both worlds: not having the time consuming problems of rewriting, but still being able to us the inference algorithms in web-applications, other programming languages and with an easy-to-use interface.

This section discovers the different options of encapsulating the existing engine in a 'wrapper'. This wrapper should hide the difficult C++ calls and enable programmers and web-developers to make use of its algorithms. Both ActiveX and the JNI-interface provide a way to accomplish this.

2.3.1 ActiveX

An ActiveX object is a special form of a COM object. Since one of the programming languages that enable you to write ActiveX controls is Visual C++, it is obvious that this will open doors to the engine written in C++. In fact it is possible to link the original SMILE engine to the code that implements the control.

By implementing the layer between the actual engine and the interfaces as defined by the ActiveX-standard it becomes possible to use the functionality of the engine in virtually every environment that supports this standard.

Like in the previous cases, the requirement of creating an easy-to-learn API can be met without considering this at this point. Appendix B.2 describes the actual implemented methods and procedures of the implementation of the ActiveX control.

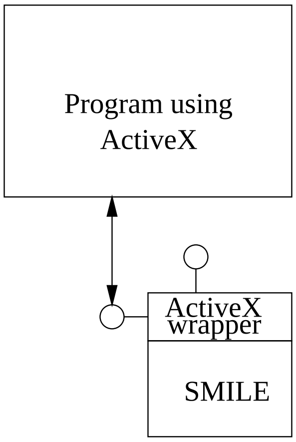


Figure 2.5: A program interacting with the ActiveX-wrapper to the SMILE library

Also the web-requirement can be met with an ActiveX implementation of SMILE. Since the Microsoft Internet Explorer supports ActiveX-objects it can be used to browse through pages containing these controls.

The same goes for other programming environments. As long as these environments support ActiveX object they will accept and be able to interact with the ActiveX version of SMILE.

2.3.2 Java Native Interface

In a way similar to the ActiveX implementation we can make a wrapper in Java. However, this wrapper has to wrap up the C++ SMILE library , so we have to make use of the *JNI* (short for *Java Native Interface*).

The JNI requires the C++ library to be compiled for the platform on which the actual Java application will run. This is somewhat awkward, since Java is said to be fully platform independent. On the other hand we can not expect implementations of Java compilers to understand or use a library compiled for any platform. In other words: it is possible to use the C++ library from Java programs without having to rewrite the whole thing, but it results in platform-dependent code.

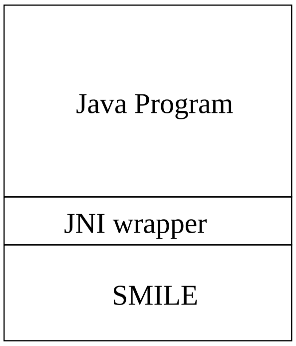


Figure 2.6: A program interacting with the JNI -wrapper to the SMILE library

By using the Java-Applet technique it is possible to incorporate a Java module into a web-page. A relativly simple change makes the engine available as web-module. One great disadvantage however is that it is virtually impossible to support web-content for all different platforms. When we concentrate on the Windows platform (as we do with ActiveX) it is quite easy to create a JNI-based SMILE version.

2.4 Dealing with the web-requirement

As stated in the problem description there is a need to make the engine available for web-applications. Until now there are two main approaches to establish this. This section will describe the two and list their advantges and disadvantages. We will start with discussing the classic client/server model followed by a description of the ActiveX model.

2.4.1 Old client-server model

In the classic approach there is a simple scheme of sending information over the network. First the client requests a page and the server sends back an interactive form which is then filled out by the user on the client machine. Then the client sends the information entered on the form to the server, which processes this information. Finally the resulting information is sent back to the client.

A scheme like this is, for example, very useable for an Internet search engine. The user surfs to a site with a search engine and fills out the form (in this case he enters a search query). This query will be sent back to the search engine, that performs the specified search. The search results are sent back to the client machine.

The bottomline in this scheme is that the client machine has no other function than to play the front-end of the search engine, the 'real work' is done on the server machine. Another point is that the query-text is sent over the network, which may be a problem when privacy factors are concerned.

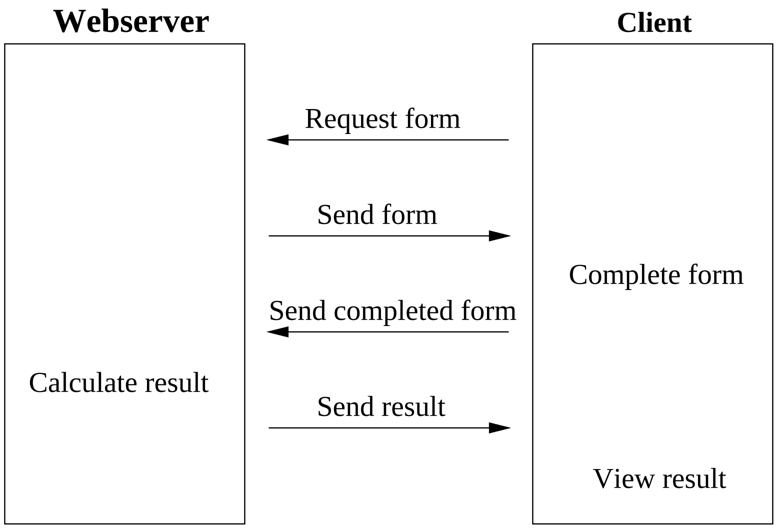


Figure 2.7: The classic Client-Server approach

2.4.2 ActiveX/JNI model

By wrapping the SMILE engine with an ActiveX or JNI layer it becomes possible to do the actual work on the client machine. The scheme is as follows:

First the client requests a form that is sent to him by the server that also sends the encapsulated engine with it. The client fills out the form, but instead of sending the information back to the server, the data is processed by the engine that came with the form. The result can be displayed directly without any further contact with the server machine.

An example: A user wants to convert his temperature measurements from Fahrenheit to Celcius. He surfs to a site that contains a conversion calculator that is immediately sent to his computer. Without any other network traffic, the user can now convert his data with an interactive form, containing the conversion module.

In this case the processing of the data is done on the client machine. Also, no network traffic is needed after the initial downloading of the module and the form. This is good if there is confidential information that has to be entered in the form (e.g. personal details in a credit application form).

One disadvantage of this scheme is that the whole module has to be downloaded to the client machine. Current developments have resulted in techniques that only require the module to be downloaded once, instead of every time a page with (the same) built-in module is requested.

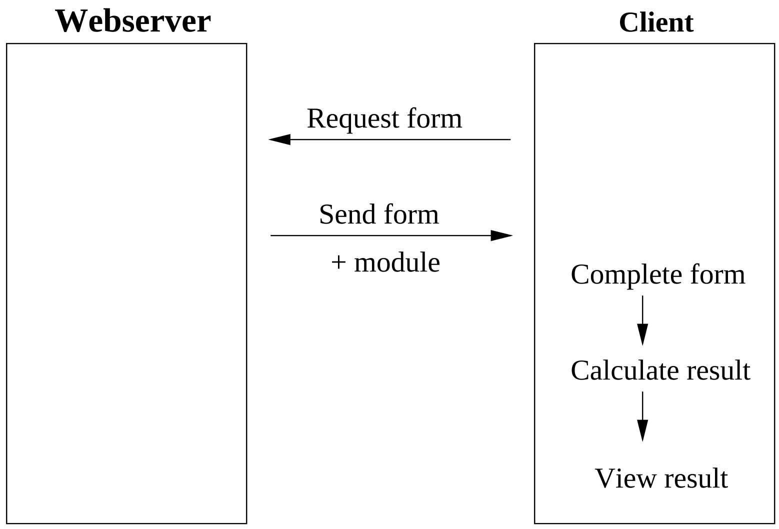


Figure 2.8: ActiveX/JNI approach

2.5 Making the choice

Clearly, rewriting the control is not an option, since this would take far too much time. Furthermore, it would result in two different versions of the same piece of software. Historically multi-version software is more difficult to maintain and support than only one version. Rewriting *would* help making the SMILE engine useable for a larger number of programmers.

Independent of the choice that is made regarding the implementation environment of the module, one can define the Application Programmers Interface to the module in a later stage. Both approaches (JNI and ActiveX) require the implementation of a shell around the original library. This shell can also serve the purpose of making the API easier to use than the API of the library. In other words, we will concern ourselves with this Programmers Interface later and concentrate on the choice between Java and ActiveX for now.

So, we are still stuck with the choice between ActiveX and the JNI interface. The final decision to go head and implement the wrapper as an ActiveX control can be founded on the following points.

- The JNI interface does not bring us the platform independence we would like to have with Java. For every platform the Java Native Interface has to be recompiled.
- By far the largest part of the userbase of the software developed at the DSL uses a Windows platform.
- As we will see in the next chapter, there is a broader set of software packages that support ActiveX controls than web-browsers only. For instance Excel and Word also support these controls.
- An important part of programmers that we want to enable to make applications based on decision support programs in Visual Basic.

Chapter 3

ActiveX, Details and Implementation

This chapter describes the details that using the ActiveX techniques imply. Besides a description of the history and theory of ActiveX some implementation details are discussed.

3.1 COM, the Component Object Model

The Component Object Model (or COM) is a technique, or rather a programming paradigm. It defines a set of rules that make sure that objects can be used from different sorts of programming languages and applications.

3.1.1 History of COM

In the old days, the only way to reuse software was by sharing source code. Sharing source code has one major disadvantage though; it can result in a lot of different versions of the same code. This is not necessarily bad, but think of the situation in which a programmer gives his sourcecode to his collegue. Say, this collegue finds a bug and fixes it only (in the borrowed code) in his own program. This results in two different versions of the original code. When this code is shared with a friend who builds some extra error-handling routines into this code, without notifying any of the other two about it, we are up to three versions of the same code.

In order to solve these problems, a mechanism was invented to make sure that people sharing the source-code could not alter this source-code. This mechanism is embodied by so-called libraries. The functions implemented in these libraries can be called from the sourcecode of the programmer. A linker will make sure that these function-calls are resolved and that the resulting program makes use of these functions correctly. If you want to share your library with a collegue, you give him the (compiled, binary) file and a description of what kind of functions you implement in that library. The collegue can not alter functions in the library and has to report bugs to you in order to have them fixed. The drawback of using libraries is obvious; if you receive an updated library from someone, you have to recompile and relink your programs that use that library. As you may recall, SMILE is available as a library.

A way to work around this problem was invented by means of Dynamic Link Libraries (DLL). These (special kinds of) libraries are packaged in some kind of executable format. In short you can say that, whenever you want to use a function provided by the the library, you load the DLL into the memory of the computer and ask it to execute the function you want to use. This solves the problem of recompiling (a new DLL can just be copied over the old one), but introduces another problem: version control. If a new version of a library is copied over the old one, this may break software depending on the older version of the DLL. An easy way to work around this problem is, instead of replacing the old library, adding the new library and give it a new name. This has a disadvantage however: old software, that uses the DLL, will not benefit from, for example, bugfixes, that appear in the new DLL, but have not been fixed in the old DLL.

In COM this problem is fixed. By giving all functions in the DLL a (globally) unique name and introducing some rules, it is possible to create a dynamic library that can be replaced by a new library without breaking and old client, and, at the same time, providing new and improved functions. By giving all functions a unique name, the problem of breaking old software is solved: If a function has the same name in a new version of a library, you can be sure that it still has the same functionality and it will be used correctly by the old software. Although the executed code has changed (maybe better performance or bug-fixes), it still has the same (format of) results, that can be processed by the so-called 'client'.

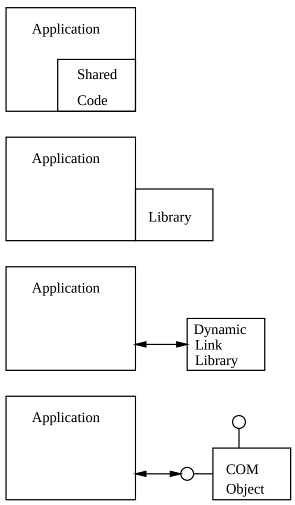


Figure 3.1: From shared source to COM

3.2 Details of COM

This section describes some details of COM that have impact on the implementation of SmileX.

3.2.1 Interfaces

As mentioned earlier COM object communicate with other processes by means of interfaces. These interfaces can be seen as some kind of *handles* or *pointers* that give processes access to the properties and methods of the COM object.

Not every method and property have a separate interface. One interface can accomodate more methods and properties at the same time. An interface can be queried to find out which methods an properties are supported by that interface.

One interface is special: `IUnknown`. This interface is the standard interface that every COM object has to support. The COM specification also requires this `IUnknown` interface to support the three standard methods `QueryInterface`, `AddRef` and `Release`. The methods `QueryInterface` enables the user of the object to find out what methods the interface supports.

The methods `AddRef` and `Release` are for reference counting. A COM object will stay in memory to handle requests from processes while there are more than zero references to at least one interface. Processes are supposed to add a reference to an interface whenever they need it and release it again when they are done.

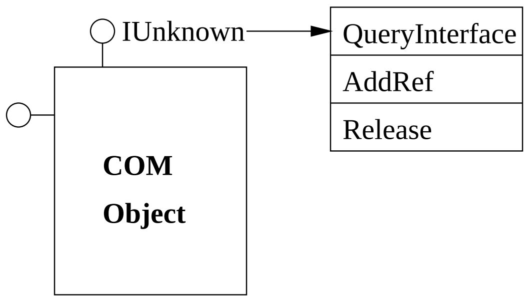


Figure 3.2: IUnknown interface of a COM object

All other interfaces are descendents of the `IUnknown` interface. As a result they automatically inherit the standard methods.

3.2.2 Unique Identifiers and the Registry

Every COM object has a unique identifier. This number is used to distinguish between the different COM objects and their interfaces. In fact, also every interface has a unique identifier.

Both the interface and object identifiers are globally unique and are composed of the network number of the computer they were developed on, the time of day and a random number. These *GUID's* (or *Globally Unique Identifiers*) are placed in the registry.

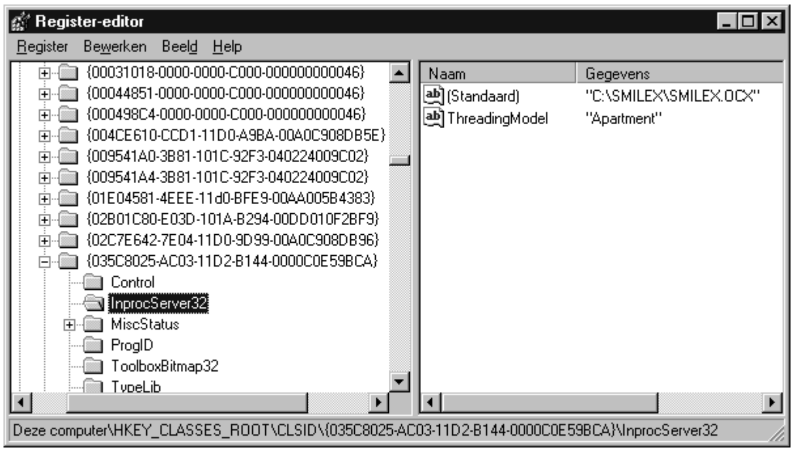


Figure 3.3: SmileX registrered in the registry

The registry can be seen as the set of configuration variables of a computer system. When a program wants to use a certain COM object it asks the registry if the object with the specified *GUID* is available. If so the system loads the object into memory (the registry also holds the place where the object can be found) and the requesting program can start using it.

The GUID for the SmileX object is 035C8025-AC03-11D2-B144-0000C0E59BCA. A good example of the use of this identifier is the way to insert a reference to the control in a web-page:

```
<OBJECT
   CLASSID="CLSID:035C8025-AC03-11D2-B144-0000C0E59BCA"
   CODEBASE="http://www.whatever.com/credit/SmileX.ocx"
   ID=SmileX
   HEIGHT=0
   WIDTH=0>
 </OBJECT>
```

3.3 ActiveX as an extension to COM

ActiveX hides a lot of the implementation details of COM. By requiring a set of standard interfaces that every ActiveX control must support, it becomes possible to have programs of different kinds interact with the ActiveX control in a standard way. The definition of this standard way makes sure that ActiveX control can be accessed from different sorts of languages, environemnts and applications.

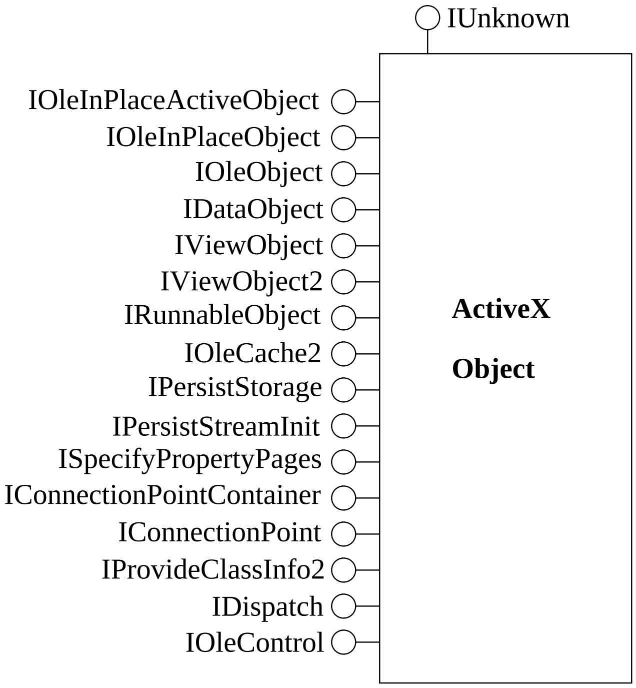


Figure 3.4: standard COM interfaces for an ActiveX control

The most important interface of an ActiveX control is the `IDispatch` interface. This interface holds the key to all methods and properties implemented in the control. Programs using ActiveX controls automatically refer to the `IDispatch` interface, that 'dispatches' the requests for properties and methods. The methods and properties of SmileX are also handled by the `IDispatch` interface of the control.

3.3.1 Variants

In Visual Basic Script and Visual Basic for Applications all variables have the same type: Variant. This datatype is a union of variables of all different supported types. Among these types are: Integers, Doubles, Characters, Strings, Booleans etc. One can think of a typedefinition of a variant as follows.

```
tyepdef struct tagVARIANT {
  union {
     struct __tagVARIANT {
        VARTYPE vt;
        WORD    wReserved1 ;
        WORD    wReserved2 ;
        WORD    wReserved3 ;
        union {
          LONG       lVal ;  // VT_I4
          BYTE       bVal ;  // VT_UI4
```

```
        SHORT      iVal ;  // VT_I2
        BSTR       bstrVal ; // VT_BSTR
        .
        .
        SAFEARRAY* parray ; // VT_ARRAY
        .
        .
            } __VARIANT_NAME_3 ;
     } __VARIANT_NAME_2 ;
     DECIMAL decVal ;
   } __VARIANT_NAME_1 ;
} VARIANT ;
```

So, a variant is a union type of all possible types a variable can have. Since a variable can also be an array of values this is also supported. An array in Visual Basic or Visual Basic for Applications is automatically an array of variables with type Variant.

VBScript and Visual Basic for Applicaton handle typeconversions internally. For instance, when a number is output to the screen it is converted first to a string and then displayed. Also when a short integer is multiplied by a long integer, the result will be of the long integer type. The field VARTYPE of the variant holds the current type of the variable in the variant and thus which field of the union holds a valid value.

This format of variables has a lot of impact on design and implementation issues, as we will see later. For example, all parameters and result values of functions have to be read as and stored in variants. In fact, a lot of the code of the developed software concerns wrapping and unwrapping variables in these datastructures.

The ActiveX control is implemented in C++ because we want to use the C++ inference engine SMILE. We have to support the Visual Basic variants in the ActiveX control. So it has to 'translate' the parameters to regular C++ types in order to use them as parameters to calls to the original SMILE engine. Also the function result values obtained from calls to SMILE have to be translated back to variants in order to be able to return them to a Visal Basic program.

3.4 Advantages and disadvantages

This sections covers the advantages and disadvantages of using ActiveX. Most of these have to do with the programming environments that support ActiveX and the implementation details.

3.4.1 Advantages

Programming environments One of the biggest advantages is that ActiveX is heavily supported by other popular software. A vast set of programming environments can make use of ActiveX-controls, thereby offering programmers an easy way to include and use software components in their software. What follows is a list of programming environments in which the programmer can insert ActiveX-components into their projects.

- Microsoft Visual C++
- Microsoft Visual Basic
- Microsoft Visual J++
- Borland Delphi
- Borland C++ builder

Note that all these programming environments run under any Windows environment: Windows 95, Windows 98, Windows NT 3.51 and Windows NT 4.0.

Scripting ActiveX-controls The beauty of ActiveX-components is that they are not nessecarily to be used in a programming environment. It is also possible to put an ActiveX-object into a document. One can think for example of a text document that needs some kind of special statistical graph in it, together with a table of the data-points shown in the graph. Suppose that we have a statistical program that can produce such graphs. Everytime that we want to change the data in the graph, we have to update the table in the document and run the mathematical program on the new dataset.

If the mathematical program would have been implemented as an ActiveX-control, we can directly access the data in the table in the document and feed the datapoints into the graph-drawing routine. The graph will be able to update automatically after we change some values in this table.

Some applications that support ActiveX object in documents are:

- Microsoft Word
- Microsoft Excel
- Microsoft Access
- Microsoft Internet Explorer
- Netscape Navigator (with ActiveX plugin)

Creating ActiveX-controls It is also important to notice that ActiveX controls can be created in more than one language. Luckily it was obvious that we had to choose the Visual C++ language for SmileX, in order to be able to use the SMILE C++ library.

- Visual Basic
- Visual C++
- Visual J++
- Borland Delphi

3.4.2 Disadvantages

The main disadvantage of ActiveX is that it is highly platform dependent. Only programs/environments that run on a flavor of Microsoft Windows can offer support and/or use ActiveX technology. Not only is this bad news for users of other operating systems (like Linux, UNIX, MAC-OS etc.), but it also means that users/developers of ActiveXcontrols are highly tied to Microsoft products. This platform-dependency also means that it is not possible to use ActiveX objects in a platform independent Java program. Only Java-programs that run on a platform that supports ActiveX-technology will be able to access ActiveX-controls.

Some projects to support ActiveX technology under the various flavors of UNIX are underway. By the time of this writing, there is no information however about the exact status or the specific details of these projects.

3.5 SMILE as ActiveX-control

SMILE is a C++ library to use Bayesian networks and influence diagrams. All programs that use these AI functionality have to be recompiled if a new version of this library is released. By making use of the COM -technology on which ActiveX is based, we can make sure that there is no need to do this anymore.

By using this ActiveX programming paradigm, it also becomes possible to use these AI functions in programs that can be written in a number of different programming environments (Visual C++, Visual Basic, Delphi) and in this way enable more programmers to make their programs work with the functionality offered by SMILE.

Also ActiveX makes it easy to make documents 'AI-enabled'. By inserting the ActiveX-control based on SMILE into a HTML, Word or Excel document, you will be able to manipulate information in these documents using SMILE-functions.

3.5.1 Using VB, VB for applications, VBScript and Javascript

In appendix B the usage of the SmileX ActiveX control is fully described. In this section some brief examples are described on how to interact with the control in order to extract decision support information from it.

First of all we need to load the Bayesian network or the influence diagram into the ActiveX control. This is done with the following statement:

```
SmileX.ReadWebFile,"http://www.sis.pitt.edu/~dsl/credit.dsl");
```

Next we set the evidence. This evidence is either obtained from a form on a web-page or from some other source. The following example sets the evidence for the node *Occupation* to whatever option a user has chosen in the combobox *MyComboBox*:

```
SmileX.SetEvidence("Occupation",MyComboBox.Value)
```

When all the proper evidence is set we need to perform inference on the current network:

```
SmileX.UpdateBeliefs("Lauritzen")
```

This uses the Lauritzen algorithm described by Huang et. al. [11] to perform inference on the network. When the beliefs are updated we have to obtain the resulting values in order to make a decision. The following statement finds the chance that the node *Profitability* is *Yes*.

```
P = SmileX.GetBeliefForOutcome("Profitability","Yes")
```

The obtained value is assigned to the variable P. Dependent on the value of P we can make a decision.

3.6 What to implement

As described in section 2.1 there were some restrictions to the module. First, it is supposed to be very easy to use and learn, and second it has to be as small as possible.

In order to accomplish this, a few things can be done. Since the size of an ActiveX control depends on the number of interfaces it offers, it is clear that we should pursuit implementation of as little functions as possible. Not implementing all functions available in the library has another advantage: the programmer that uses the API does not have to remember so much functions or browse through pages or books full of documentation about the API. This surely accomodates the requirement to make the module easy to use.

Overmore implementing a wrapper function for all methods and properties of the original SMILE library would be too time-consuming and error-prone. In that case it would be more advisable to rewrite the entire engine.
Application programmers, using probabilistic reasoning in their software, will not often need to build or modify a model. In general, the model will already be final, and can thus be created independently of the application. One can for example think of the proprietary interface for diagnosing liver disorders. The model was built in GeNIe, and the application provides a user-interface to the model that is easy to understand for the users.

When making this observation, it is obvious that this is a good opportunity to leave a great number of functions (model building and modifying functions) out of the API. This implies however that the end-user-application has to create the model using GeNIe to build the network. Since GeNIe is especially designed to build and modify networks, this implication does not have to be considered as a drawback.

3.7 Applications with SmileX

Using the inference algorithms has become easier for developers of web-sites, end-user applications etc. This section presents two examples which are based on the SmileX ActiveX control.

3.7.1 Hazardous materials

One of the projects in the class concentrated on predicting the type of hazardous chemical, given some properties of these chemicals. In case of an emergency one can find out quickly with what kind of chemical one is dealing.

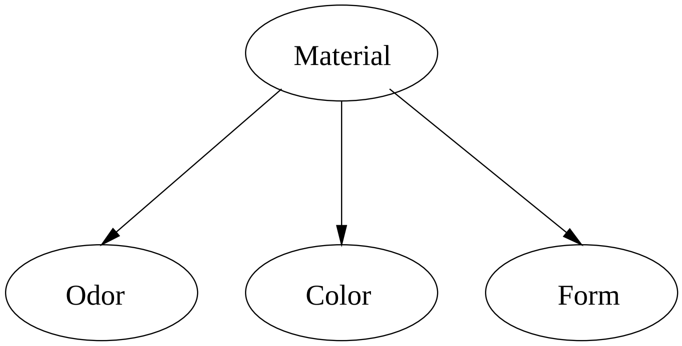


Figure 3.5: Model for prediciting type of chemical

The model is actually quite simple. The three properties are influencing the type of material directly. In the final model 35 materials were listed with 15 different odors (Garlic, Rotten cabbage etc.), 13 different colors and three possible forms (liquid, gas or solid).
The application was implemented as a web-page. By using the SmileX control and Visual Basic script it was possible to have the user choose between the different properties and combinations thereof. After changing a property (with the comboboxes on the left) the current evidence is set and the new chances are computed and displayed on the right. Of course influence algorithms in SmileX are used to compute these chances.

3.7.2 Lawfirm

This application uses a model that describes the decision process of a lawfirm where a choice has to be made whether or not to take on a case. By entering information in the model (by means of the Graphical User Interface) one can calulate whether it is worth it to take the case. Besides profit, the reputation of the lawfirm and the chance of winning the case play a big role in this decision process.

This interface was created under the Microsoft Visual Basic programming environment. It inserts the SmileX ActiveX control on a form in order to communicate with the model that was built to describe the decision process. The model is quite complex, but the GUI makes it easy to set the evidence and calculate

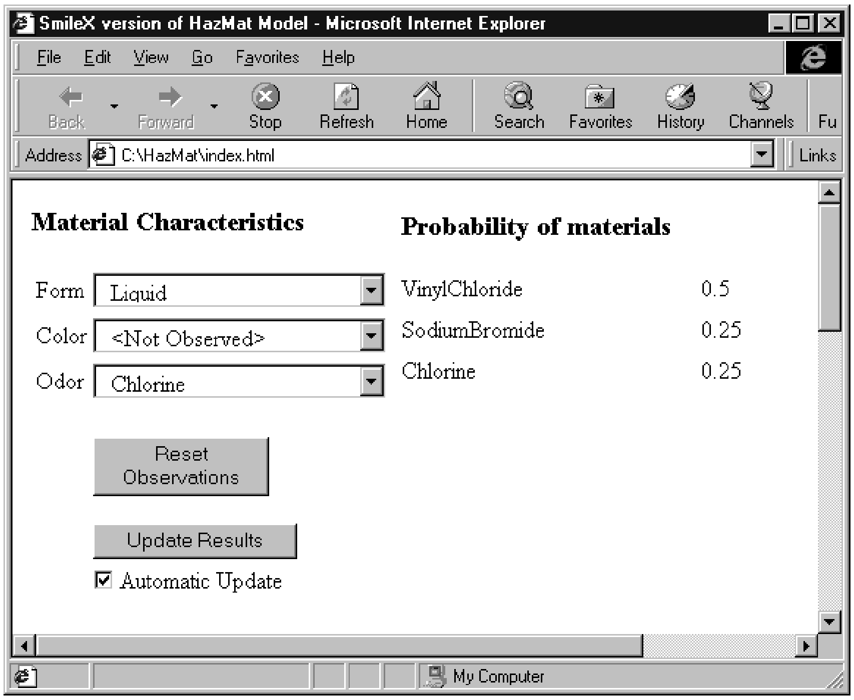


Figure 3.6: WWW interface for hazardous material application

the results. This way the it can be used by people that have no knowledge about Bayesian networks or inference algortihms whatsoever.

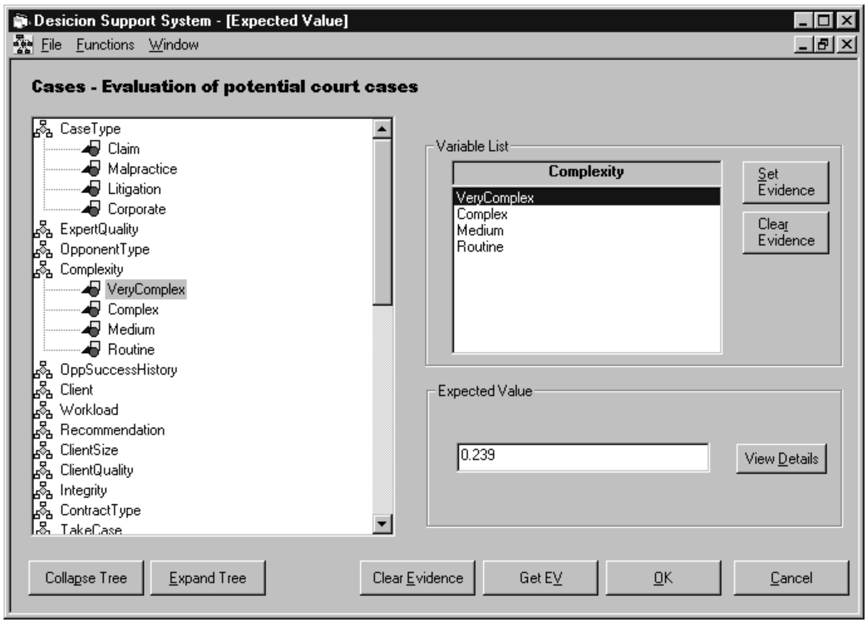


Figure 3.7: Visual Basic interface for lawfirm application

Part II

Credit Evaluation in Consumer Credit

Chapter 4

Problem description and environment

This chapter sets the scene for the second part of the report. It contains a description of the credit-evaluation problem that will be subject to research in section 1. Section 2 gives some insight in the ways this problem has been researched and is dealt with and section 3 delves into the details of the approach taken in the course of this project.

4.1 Problem description

This report proposes a model that can help in evaluating creditworthiness in commercial consumer credit. This section delves into the conditions and specific requirements of the project.

4.1.1 Type of credit

There are several types of consumer credit. For a fairly complete discussion of these types, the reader is advised to read Nunlist [15]. The type of consumer credit that is discussed in the rest of this report is the so-called *consumer installment credit*. This type of credit was developed according to consumer needs, to enable the customer of the bank to buy a car, an expensive appliance or something else, without having to pay for it up front. Nowadays the credit does not especially have to be used to buy something large, it can also be used for something else, like an education or to raise children. The main procedure (obtaining a large sum of money and paying it back in monthly payments) resides however.

4.1.2 Purpose of loan

In present-day banking it is usually the case that the amounts in consumer installment credit is not more than $75,000 to $100,000. These amounts will be enough to buy a fairly expensive car, a small house or an apartment. The purpose of the loan will remain an open issue, thus it can also be used for refinancing and consolidation. Some of the most common purposes for applying for a loan are:

- Car
- Education
- Raising children
- Refinancing
- Appliance (Stove, refridgerator, washing machine etc.)
- Housing (Usuallly a mortgage is preferred over installment credit)

In this report we consider fairly large commercial banks. One can think of PNC and Mellon Bank in Pittsburgh, PA) for instance. Of course a lot of the information in this report and a lot of ideas in the model also apply to other lenders.

It should be noted here that this whole research project has been done with banks in the United States of America in mind. Since the USA has different laws, rules and customs than other countries this model can not be used in other countries without modifications.

4.2 Current approaches

4.2.1 Human judgment

The process

These days, most banks employ people to make the decisions wether to grant or deny the loan applications. Typically, the first step to be taken to apply for a loan is to fill out an application form. On this form the applicant has to fill out a lot of information about himself, his income, expenses and so forth.

The second step is usually an interview of the applicant conducted by a loan officer of the bank. This loan officer usually goes through the application form with the customer and ask him some more background information about the particular case at hand. Since most application forms are very generic it is often only in the interview with the customer when special cases or conditions will be identified. The loan officer will also form a personal opinion about the customer and his case in this step. The loan officer will also verify certain facts and claims of the client (like educational background, professional status etc.) by asking for references and evidence.

The next (third) step will in general be conducted by a risk analyst that calculates things like the relationship between the expenses and the income of the customer, the total profit that the bank will make by issuing this loan, a statistical analysis of comparable cases etc. Usually this stage has little or nothing to do with the the evaluation of the character or special circumstances of the customer, it is rather a general number-crunching process. In most cases a background check on the credit history of the client is also done by the risk analyst.

The final step in evaluating creditworthiness in commercial consumer credit is taking the actual decision to either grant or deny the loan. This decision is often made based upon the recommendations of the involved loan officer and risk analyst. Usually a loan committee or senior loan officer take the decision and communicate the result to the customer.

Disadvantages of Human Judgement

The following disadvantages can be identified:

Complexity The actual decision is difficult to make. This is due to the complexity of the problem. The decision depends on a lot of variables that all have to be taken into account at the same time.

Inconsistency Since the recommendations and all intermediate decisions are made by humans, they are prone to be subject to moods and personal preferences of the humans involved. This means that the final decision can for instance partially or entirely depend on the mood of the loan officer.

Inconvenience The customer applying for a loan may find this process highly inconvenient. It typically requires two or more trips to the bank, a waiting period and an interview before they have an idea whether they have any chance to be granted a loan.

Expensive Evaluating credit applications id a very time-consuming task. Every application has to be evaluated carefully to come to the right decision. This takes up a lot of human resources and thus a lot of money.

Advantages of Human Judgement

One of the big advantages of conducting the interview with the applicant is that it is very easy for the client to express any special circumstances that can not be brought to attention by filling out an application form.

Also, some customers may appreciate the personal interaction with a bankemployee, helping him to figure out the best way to apply for a loan.

4.2.2 Judgement based on Neural Networks

Fundamental ideas of neural networks

The human brain supposedly consists of a network of neurons that interact with each other in order to serve a higher purpose (e.g. to come to a decision). A neural network is modeled after these processes. Depending on the interactions (by means of tiny electric currences) neurons will react by interacting with other neurons.

In computer science, a neural network works similar. A computer program emulates the neurons and their interactions. The big difference however lies in the (current) input/output systems of these neural networks. In some way, input parameters have to be fed into the network. A conversion has to be made to translate the parameters into a certain configuration or states of neurons. To generate output that actually makes sense, the resulting state of the neurons has to be translated back to result values.

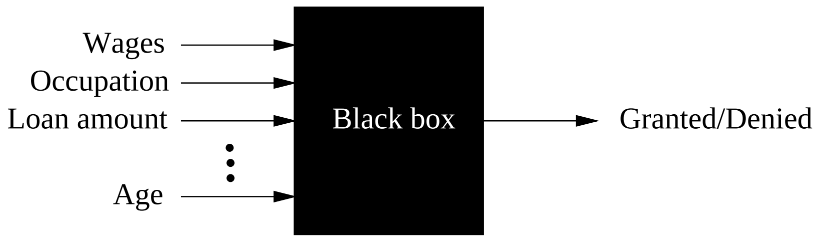


Figure 4.1: The credit evaluation problem using neural networks

Implementing a creditworthiness evaluation system as a neural network

In 1996 Glorfeld and Hardgrave [10] have developed a neural network model for evaluating credit applications. In this study twelve input variables (loan amount, loan duration, current assets, income etc.) and two output values ('granted' or 'denied') were used.

Two different kinds of neural networks were developed: a single-layer network and a multilayer, feedforward model using the very common backpropagation learning algorithm. A model considering all input variables is optimal, however reducing the number of variables can help to simplify the model. By dropping the variables with the least impact on the outcome first, we can keep the accuracy on the same level as the optimal model and simplify it at the same time.

The networks were trained by splitting the input in 10 training sets and 10 validations sets. All sets contain 50 cases. By exchanging cases from these sets one at a time and running the learning and validation algorithms again, we can get a better model. This technique is known as *cross-validation*.

A single-layer network using 5 input variables (total assets, total liabilities, net worth, total direct debt and loan amount) turned out to be equivalent to the optimal model, giving a performance of 75% correct-classification.

Advantages

Machine evaluation The key advantage of using a neural network to help deciding is that it is done by a machine. Therefore, the recommendations are not subject to human moods and preferences.

Easy to learn It is fairly easy to learn a neural network, that is, if enough data is at hand to make a consistent network.

Model There is no need to build a model to represent the decision process. This means that little domain knowledge is needed to make a neural network for this decision problem.

Disadvantages

Testcases Neural networks are dependent on learning. This leaning stage, also called training stage, requires known cases to be fed into the network together with their (supposedly correct) results. Banks would have to feed cases into the model that have successfully been evaluated by manual judgement. One of the problems that arise from this issue is, that it is not known whether cases, in which credit was denied, would have turned out to be non-profitable indeed. Only cases that were evaluated as profitable and resulted in a profit can be used as learning material.

Black box It is virtually impossible to determine why the neural network has generated a certain result. In this case this is not acceptable, since customers are entitled to hear why they have been denied credit.

Special cases Cases in which there are special circumstances will generate random results. Especially if the training sets do not contain similar cases.

4.3 Bayesian Network approach

In this project we have chosen for the Bayesian network approach. The most important reason for this choice is the availability of expertise and knowledge in this field. In the DSL a lot of research and development is done on this subject and it is the perfect environment for a project on evaluating creditworthiness based on Bayesian network theory.

4.3.1 Disadvantages

Number definition values The key disadvantage of using a Bayesian network to model the uncertainty in evaluating consumer credit is the amount of numbers that are required to define the model. For every variable that is influenced by other variables a number of values must be given that define the measure of influence in different cases. When a network has many variables the number of definition values can grow quite large (The current model contains more than 1000 values). Of course these values have to make sense, so they have to be carefully researched and entered. When test-data is available these definition values can be learned from this data however.

Requires theoretical and expert knowledge at the same time Often, when a Bayesian network has to be created, this is done by more than one person. Often a team is composed of one or more specialists in Bayesian network theory and one or more specialists on the subject of the decision. This means that time of more than one person has to be allocated in order to come to a requisite model. Besides time (and thus money) issues it is sometimes difficult for a group of people to communicate their idea's and views on things to eachother.

4.3.2 Advantages

Insight in the decision process By using a Bayesian network to model the decision you want to make you are forced to make a representation of the environment in which the decision has to be taken. It requires you to think hard about the different issues and variables that influence the decision either directly or indirectly.

If not much, it will give you some insight about the problem in general and in the way the decision is to be taken. In some cases modeling problems with Bayesian networks can even give insight about the way in which the decision makers *like to* make the decision.

Not dependent on learning As opposed to neural network models a Bayesian network is not dependent on learning. Although learning can be used to help obtain the definition values, a set of sample cases is not required in order to generate results.

4.4 Problem Analysis

4.4.1 Objectives

Since we are talking about commercial consumer credit we can define the primary objective for a bank as *making profit.* In order to gain as much profit as possible we can come up with some sub-objectives that must be addressed in this particular case.

A good place to start is the sub-objective of successfully screening the loan applications. After identifying a case that has a high chance of being profitatble to the bank, it is easy to choose wether to grant a loan or not. Obviously, a loan that can not be paid back in full by a customer (e.g. due to a bankrupcy) is less profitable than a loan that is paid back as agreed.

On the other hand, the bank wants a happy clientele, which also does good to the amount of profit the bank makes. In order to acchieve this, it is nescessary to be customer service-oriented, which can be seen as the second sub-directive. If the bank would only grant loans that have a *very high* probability of being profitable, this would result in a smaller customer-base. Customers that applied for a loan with 'only' a *fairly high* probability of being profitable would be denied credit. However, the competition would gladly grant them a loan and consequently take away customers (and profit) from the other bank.

When these two sub-objectives are considered one at a time, there is no problem. The thing is however, that these two objectives have to be seen in combination. The bank has to find some way to weigh these two factors against eachother

in such a way that the amount of profit will be maximized. When two (sub-) objectives have the opposite effect on a decision, they are called *conflicting objectives*.
Another set of conflicting objectives can be found in the fact that banks want to collect as much information about the loan-applications and the applicants as possible, in order to base the outcome on as much fact as possible. If the bank has found that 90% of home-owners have paid back all their due amounts in time, it is profitable to partly base the decision to grant or deny a loan on this fact. The information the bank needs from the applicant is whether or not he owns a house.

It is illegal however to base the result of the decision process entirely on e.g. the applicant's race or sex. Since lawsuits have a negative effect on both the profit and the good reputation of the bank, it is better to avoid them and to evaluate applicants according to the applicable laws (see section 5.2).

4.4.2 Players

The following is a list of players involved with the decision process of of evaluation commercial consumer credit applications:

Lenders In short, this group can be thought of as the bank that is trying to evaluate the loan application. Nowadays a lot of financial institutions offer consumer credit. This project focuses on credit offered by fairly large banks. A lot of information in this report can also apply to other lenders however. It is the lender that issues or denies the loan, based on the opinion of its decision makers.

Consumers The consumer is the actual applicant. It is the customer of the bank that wants the loan for his own use (which includes spouse and children) In order to be granted a loan, he has to convince the decision makers that his application is profitable for the bank.

Decision makers Every decision needs one or more decision makers. A typical lender has often one or more of following employee(s) that are responsible for making the decision:

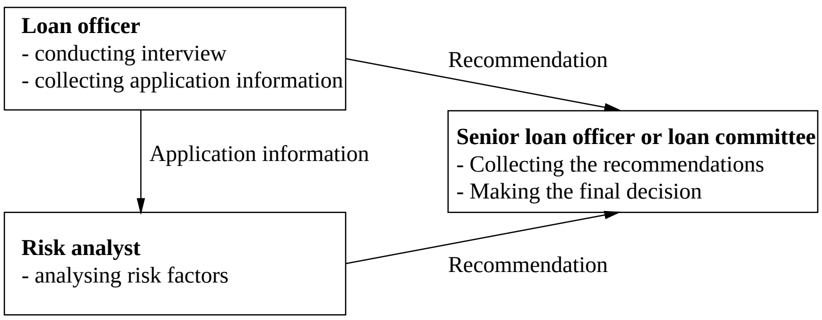


Figure 4.2: Players in the decision making process

- **Loan officer** Usually responsible for conducting an interview (for character evaluation of the customer) and acquiring the appropriate information from the client on which the other decision makers base the decision.
- **Risk Analyst** Responsible for analysing the risk factors involved in the decision process. The risk analyst recommends the proper action to take based on numbers. These numbers are usually statistics of old cases in combination with calculations on income/expense habits of the applicant.
- **Senior loan officer or loan committee** The senior loan officer and/or the loan committee are often the actual decision makers, basing their opinion on the recommendations of the loan officer and the risk analyst.

In many cases these groups of decision makers are not disjunct. It is for instance very common that the loan officer, that acquired the actual information about the application, takes part in the loan committee. In some cases the loan officer *is* the loan committee and takes the decisions single-handedly.

4.4.3 From Probability to Decision

The ultimate decision that has to be supported is the decision of the senior loan officer or loan committee wheter to extend credit to a customer or not. Since we can only represent probabilities in Bayesian networks we need to develop a way to extract decision support information from the network.

In short, the decision whether or not to grant a loan in a particular case depends only on the chance of profitability of the loan. This means that, in order to obtain decision support information from the network, we have to let the algorithms for updating Bayesian networks do their work and look at the probability of the loan being profitable by checking the value of the outcome *Yes* of the node *Profitable*.

This probability will range from 0 to 1. A value smaller than .5 can be explained as being too small a chance to grant the loan. Since we want to relate the result to profit it is better to substract 0.5 from the observed chance. This way we obtain a negative value for cases that have high probability of a negative profit and a positive value for cases that have high probability to result in a positive profit. Thus:

$$P(\text{Profit}) = P(\text{Profitable} = \text{Yes}) - 0.5$$

A positive and negative value for a positive and negative recommendation are more intuitive, but still say nothing about the amount of the profit that can be made (or lost) by granting the loan. The order of magnitude of this profit can be obtained by inflating the previous result by the loan amount:

$$\text{Profit} = \text{LoanAmount} * (P(\text{Profitable} = \text{Yes}) - 0.5)$$

If we actually want to estimate the profit, made by granting the loan, we should substract the Loan amount from the sum of monthly payments that will be done.

$$\text{Profit} = (\sum \text{Monthly payments} - \text{LoanAmount}) * (P(\text{Profitable} = \text{Yes}) - 0.5)$$

Of course the bank wants to make at least as much money by issuing the loan as it would make by 'doing nothing' (i.e. not granting the loan and keeping the loan amount invested in some kind of fund or stock bonds). Here we will assume that not issuing the loan will result in a profit of $n\%$ of the Loan Amount. Caluculation of the *extra profit* is as follows:

$$\begin{aligned} \text{ExtraProfit} = ((&\sum \text{Monthly payments} - \text{LoanAmount}) - \\ &((1 + \frac{n}{100}) * \text{LoanAmount})) * \\ &(P(\text{Profitable} = \text{Yes}) - 0.5) \end{aligned}$$

This gives us a pretty good estimate of the Extra profit. The sign of the value still has the meaning of a positive or negative recommendation. After all a negative extra profit would mean that the bank should keep its money and invest it something else rather than to grant the loan. A positive extra profit would mean that the bank will probably make more money by issuing the loan.

Notice that the only variable, that is highly uncertain, is the probability that the loan is profitable. This is exactly the value that is obtained by executing the Bayesian updating algorithms on the network. Keep also in mind that this uncertain variable is a probability. Thus, despite all the corrections, this formula will never produce exact results.

Chapter 5

Evaluating Creditworthiness

This chapter describes the result of the research that was done in order to obtain the variables that influence the decision to either grant or deny credit. Where this chapter will discuss general topics about the separate variables and their use, the next chapter will introduce the model that was built and the influence of the variables on the decision.

5.1 Literature

In existing literature we can find a lot on the subject of evaluating creditworthiness. Besides numerous models using different techniques there is some kind of common knowledge in this field where a lot of these models are based on.

No Bayesian network models have been done to help in evaluating creditworthiness until now. Nevertheless the details of other approaches, described in literature, have helped develop the current Bayesian model. Like other models, (e.g. the neural network model discussed in section 4.2.2, it depends on a set of variables that have to be assigned a value by the loan applicant and/or a bank-employee. After entering this information the model will evaluate the creditworthiness of the applicant.

5.1.1 The six C's

In evaluating creditworthiness it is common use to evaluate two groups of variables. The first group evaluates the possibility that the client defaults on his loan. The second evaluates the assets of the applicant. These groups of variables are usually subdivided into subgroups.

In literature there are numerous of these subdivisions known. Some studies mention more than 12 subgroups. Actually, for this Bayesian network approach it does not matter how many subgroups are defined, as long as the variables in these subgroups are taken into account in the correct way.

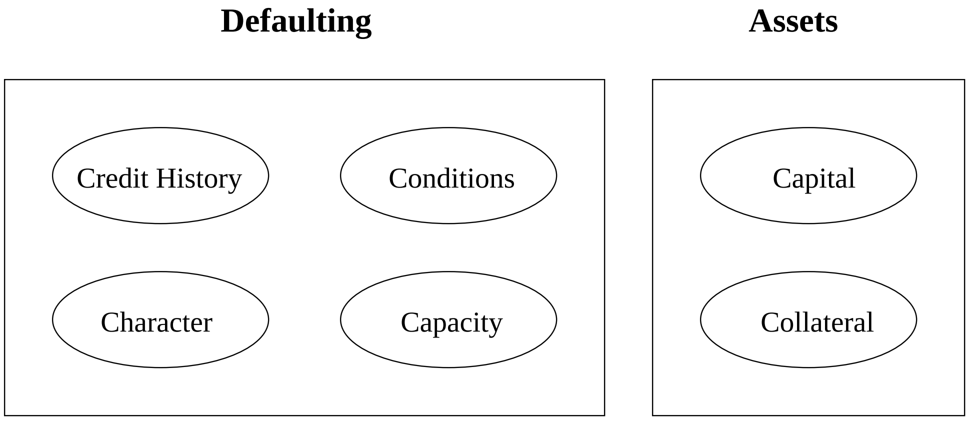


Figure 5.1: The six C's

For this project we have chosen to define 6 subgroups, this subdivision is known in literature as *the six C's*, since the names of the subgroups all start with a C. In figure 5.1 the six subgroups are displayed with their main groups. The rest of this section gives a short description of the groups.

Credit History As we will see later, the credit history of a customer is not determined by the bank itself. This information is obtained from a credit bureau that keeps records from as many people as they can. For an example credit report see figure 5.3.3.

Character Another important issue in the creditworthiness evaluation is the determination of the personal characteristics of the person applying for a loan. People with an unstable character are more unlikely to be able to pay back the due amounts in time. The character of an applicant can be evaluated by (among others) looking at the frequency of the applicant writing bad checks, the number of legal suits he has been involved in a.s.o.

Capacity In short the capacity of an applicant is the variable that determines whether the person is and will be able to pay his monthly due amounts in time, based on his income, expenses and (job-) perspectives.

Conditions Under conditions we list all the external factors, that can have influence on the loan. By external we mean that neither the bank nor the customer, that applies for the loan, will be able to influence these factors. One can think of economical depression, political fluctuations etc.

Collateral Collateral is the value of the possessions of the applicant that can be repossessed whatsoever. Usually collateral is actually owned by the bank until the loan has been paid back. This way, the bank will be ensured of the ability to sell the collateral, without the worries that other creditors of the applicant can reposess it.

Capital The capital of an applicant is that part of his belongings that can be repossed, without being owned by the bank. A good example of capital is a coin collection. It should be clear that collateral and capital are disjoint. In fact, they are the individual parts that sum up to the assets.

5.2 Legal use of the variables

By decisiding on creditworthiness one needs to make sure that the decisions are made on legal grounds. Besides deciding on legal grounds some acts have impact on the way we make the decisions in this field.

Equal Credit Opportunity Act This act disallows dicrimination of persons (and thus applications) based on sex, religion, race etc. It also grants the applicant the right to a written report of denial of the loan that states the reasons of denial. For this project this act is important because the problem has to be modeled in such a way that these variables can not seperately distinguish the approved applicants from the denied applicants.

Fair Credit Reporting Act This act establishes not only procedures for correcting mistakes on a person's credit record, it also entitles customers to a cost-free credit report within 30 days of denial of credit. This last point means that, based on the results from the model, it should be possible to generate a report with reasons for denial of credit for the specific case.

Consumer Leasing Act The consumer leasing act entitles consumers to disclosure of information that helps them compare costs and terms of various leases and the cost and terms of buying on credit versus cash. This act forces us to reveal the calculations done in the model to the client.

5.3 Acquiring information about the application

This section will cover the topic of gathering of information about the application. It will describe the different pieces of information that are acquired during the differentstages of an application.

5.3.1 The applicaton form

An important source of information about the loan is the customer application form. From most questions we can easily determine why they are asked. Some questions are less obvious however. For example, the question how many years the applicant has spent on his current job may seem somewhat weird and some people may think it has nothing to do with this application process. As we will see later all questions do have a reason.

5.3.2 Interview

During the interview the bank will acquire additional information about the environment variables of the client. It is difficult to gather this information by means of an application form, since these variables are different in every case.

A list of questions typically asked during an interview:

Applicant Information	
Last Name	
First Name	
Middle Initials	
Birth date	
Number of dependents	
Current address	
Address	
City	
State	
Coutry	
Length of residence at this address	... months ... years
Previous address (if less than 2 years)	
Address	
City	
State	
Coutry	
Length of residence at this address	... months ... years
Employer Information	
Employer	
Occupation	
Phone	
Lenght employed at this position	... months ... years
Previous employer (if less than 2 years)	
Employer	
Occupation	
Phone	
Lenght employed at this position	... months ... years
Income	
Gross Salary (per month)	
Other Income (per month) Source	
Expenses	
Rent/Mortgage	
Car payments	
Other	
Real Estate	
Date of purchase	
Purchase price	
Current Market Value	
Car	
Year	
Make	
Other assets	
Market value	
Loan	
Amount requested	
Duration requested	

Figure 5.2: A typical application form

- What is the purpose of the loan?
- Why do you want a loan for this purpose?
- What is your income and what are the prospects of your income?
- Can we expect you to pay back your monthly due amounts in time?

As you see this list contains some questions that can also be asked on an application form. What is important however is the reaction and motivation of the applicants. Overmore, the loan officer, that must translate his findings into recommendations, is given an opportunity to assess client.

5.3.3 Credit score

The credit history of a client can be obtained from a commercial credit bureau. Different credit bureaus have different systems to describe or evaluate different individuals. Usually different events in the credit history of an individual are weighed and put together to produce a score. In a way the credit bureaus also make use of a model to evaluate people and their creditworthiness.

Credit scoring system	**Score**
Beacon score	00742
Empirica score	00706
Fair Isaac score	00751

Creditor	**Date reported**	**Balance Owing**
JC Penney	08/1996	$0
Visa	08/1996	$1827
Northwest Mortgage Co.	09/1996	$13K

Figure 5.3: Example of information on credit report

5.3.4 Internal Bank Information

Usually people that are already a customer of a specific bank have a higher chance of having their loan application approved. Especially for customers that have been good customers of that bank have that chance is higher.
In fact banks check whether the applicant has been a good customer. They base their opinion (and thus the extent in which they want to be client-friendly) on the following topics:

- Whether the customer writes bad checks or not.
- Whether there is a clear regular income on the account or not.
- Whether the client has had problems with keeping up with payments.
- Whether the client has problems with his credit limit.

Some of these topics are also asked on the loan application form, but are (double-) checked anyhow.

Chapter 6

Credit Evaluation Model

This chapter presents the model that has been developed using the techniques and remarks in the two previous chapters. In the first section an overview of the model is given, followed by a presentation of the submodels. An important part of constructing a model is not only to draw a diagram, but also to add probabilistic information about the influence of variables among eachother. Section 3 will describe how these numbers were acquired and gives a quick view of possible better ways to acquire these numbers from an expert or data.

6.1 Overview

Figure 6.1 represents the overview of the credit evaluation network. It is clearly the result of applying the six C's to Bayesian network theory.

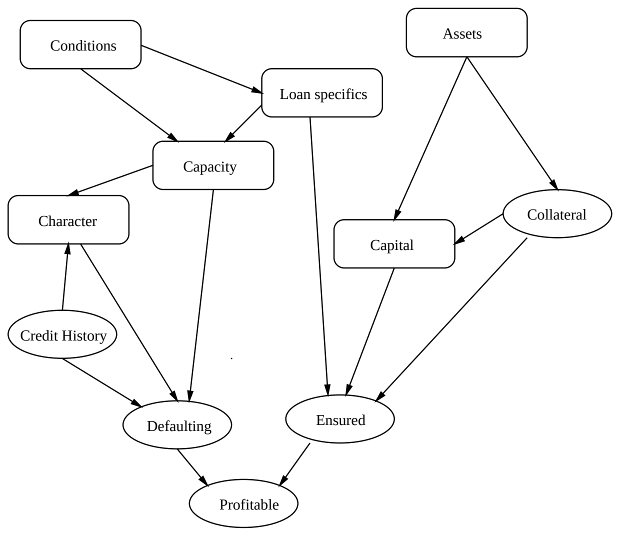


Figure 6.1: Overview of the credit evaluation model

In this model the rectangles with rounded corners are submodels. These submodels are discussed in detail in the next sections. This overview shows that the profitability depends on two factors:

1. What part of the loan is ensured
   This variable holds a percentage that corresponds with part of the loan amount value that can be reposessed in case of defaulting. In general this variable is the result of some elementary calculations.

2. Defaulting
   This variable holds the chance that a certain apllicant has to default on his payments. It is the result of applying Bayesian algorithms on the rest of the (non-numerical) part of the network.

Also, since the bank does not participate in establishing the credit score, this variable is modeled as-is. As a result it has no incoming arcs. Variables that do not really belong to a submodel, but are coming into the submodel or are exported from another submodel are drawn without borders.

6.2 Submodel conditions

The submodel conditions models the exterior influences of the environment on the decision problem. Besides political fluctuations and economic fluctuations it could for instance contain conditions like weather influences. In order to simplify the model and because there are no obvious measures for such variables, these additional conditions have not been modeled.

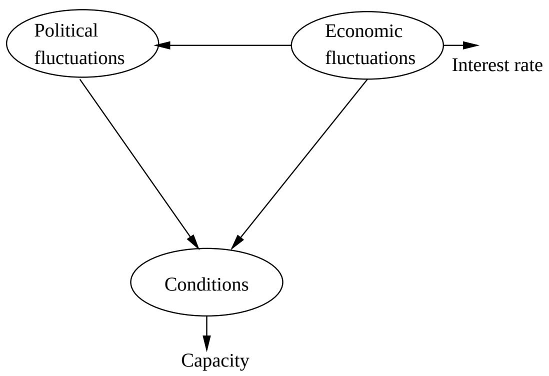


Figure 6.2: Submodel conditions

Clearly economic sluctuations have their effect on the interest rate. The accumulation of these conditions is the node *Conditions* that influences the *Capacity*.

6.3 Submodel Capacity

In this submodel the capacity of the applicant is modeled. On the righthand side the so-called *Income Debt to Ratio* is calculated. This ratio is supposed to represent the height of the monthly income of the applicant divided by the total of monthly payment commitments, like car payments, rent etc. On the left-hand side in this submodel the income prospects are evaluated, based on the type of job, the number of years at the current job, the applicants age etc.

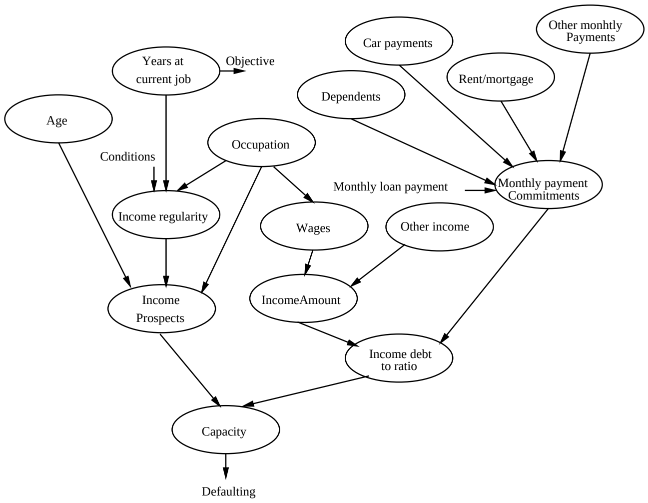


Figure 6.3: Submodel capacity

Applying the Bayesian theorems (described in section 1.3.2) on the results of the nodes *Income Debt to Ratio* and *Income prospects* we end up with the chance that the applicant will be able to pay his monthly loan payments, now and in the future.

6.4 Submodel Loan Specifics

The submodel *Loan Specifics* represents the terms and conditions of the loan. It models the relationship between the Interest rate, the Loan amount, the loan duration and the monthly payment. Also the purpose of the loan is incorporated in this submodel.

The main result is the monthly loan payment, which is 'exported' to the submodel capacity where it is added to the other monthly payment commitments.

6.5 Submodel Character

The submodel *Character* exposes the variables that are used to evaluate the personality or character of the applicant. The model is split into two character evaluations:

1. Objective
   On the left-hand side of the submodel character we evaluate the properties of a character that can easily be determined and entered into the model. Ususally there is no discussion about the applicable values of these variables in each case. The node *Objective* is the accumulation of the variables

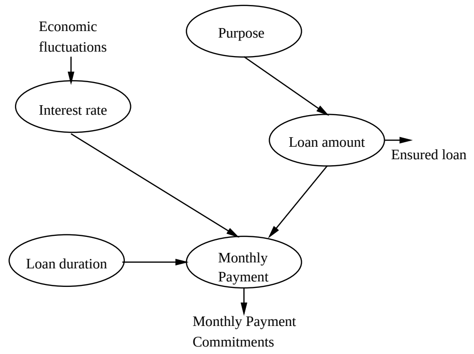


Figure 6.4: Submodel loan specifics

Years at current residence, *Years at current job* and *Honesty*. Honesty is a submodel and will be discussed in the next section.

2. Subjective
   In this subjective part of the character evaluation we model the subjective observations or properties. During the loan application interview the loan officer interviewing the applicant will make these observations. By setting evidence for these nodes the loan officer can make his opinion count.

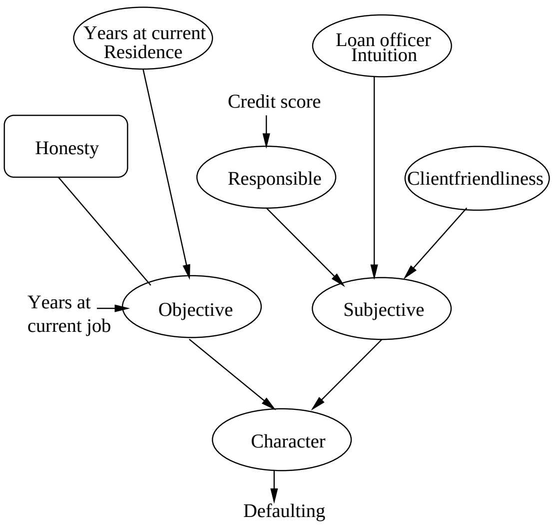


Figure 6.5: Submodel character

The complete character evaluation feeds into the node that determines the chance of defaulting. Clearly people that are not honest and make a bad impression on the interview have a bigger chance not to be able to make their monthly payments.

6.6 Submodel Honesty

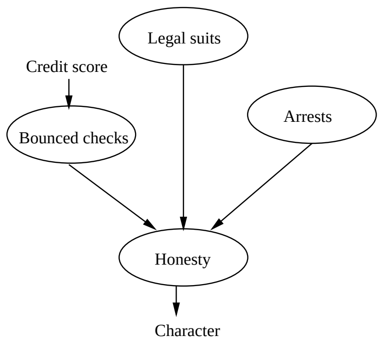


Figure 6.6: Submodel honesty

Here we try to determine the honesty of a person based on the number of legal suits, the regularity of bad checks and the number of his arrests. The number of bounced checks is influenced by the credit score.

6.7 Submodel Assets

The total assets of an applicant can be calculated by adding the total of liquid assets (like stock bonds, mutual funds etc.) to a certain fraction of the total of semi-liquid assets (car, real-estate etc.). This fraction is usually 80% and is applied to make up for the costs of selling the semiliquid assets in case of defaulting. In short the assets are the sum of money that can be obtained by selling the applicants possessions.

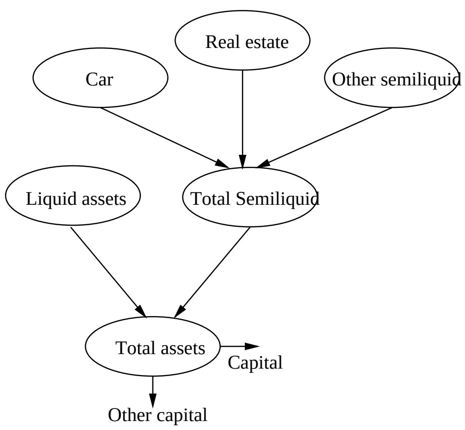


Figure 6.7: Submodel assets

A part of the assets can be put up as collateral. Collateral is the part of the assets that is actually owned by the bank until the loan is paid off. The other part of the assets is called *Other capital*.

6.8 Submodel Capital

The node *Capital* determines how much of the *Other capital* can be reposessed by other banks or lenders. It is actually the part of the assets the bank, considering to offer the applicant a loan, can have a claim on when the customer defaults and the collateral does not suffice.

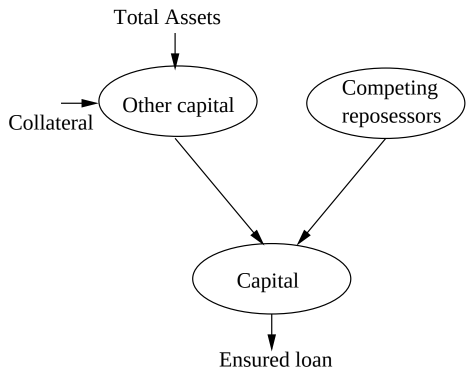


Figure 6.8: Submodel capital

6.9 Values of the model

In the remainder of this report we will talk exhaustively about different sorts of values of the model. The following list identifies these values and introduces a name for them, in order to be clear about what values we talk in the future.

Definition values The values and parameters that are part of the model, i.e. they define the model. These values include for example the chance that someone will lose his job. They are generic and are used for all applicants/cases.

Input values These values are put into the model and are obtained in order to evaluate a certain case. Among these values are for example the amount of assets and the income of the applicant. In some cases we will need to process raw input data before putting these values into the model. This is due to the fact that the current software does not support continuous variables and because it does not support simple calculations either.

Result values These are the values that get generated by the inference algorithhms used on the model. The most important result value is the value of the chance of profitability.

6.10 Acquiring the numbers

In the previsous section we talked about definition values. These values are part of the model and have to be entered in order to obtain usefull decision support information. This section describes some ways to obtain this information.

In the context of creditworthiness we have to obtain for example numbers about chances of defaulting, given that the applicant have their own house, or maybe the chance of losing a job, given that someone has been working for 20 years at the same position. Also general numbers have to be obtained, for example the age distribution of applicants.

6.10.1 General access sources

Certain numbers can be obtained by using general access sources. In literature one can find statistical data concerning for instance age distributions and income levels. A lot of this information is also available from statistical bureaus, some of which publish this data on the Internet. Usually this information is rather general and not applied to the creditworthiness problem. Nevertheless, in this project we have worked with data obtained from the census bureau. Numbers about mean income, age groups, money spent on children etc. were found on the census website at `www.cesus.gov`.

Besides statistical data one can make use of common sense. It is quite common to use figures that seem reasonable. In general we can make estimations of chances simply by do simple reasoning. For instance it is obvious that people that have worked only for 2 months in their current position have a higher chance of losing their job than people that have had the same position for more than 2 years.

Of course the bank also keeps records of their clients. For instance information about bad checks, regular income and credit/debet status is recorded. The information in these databases however, only applies to current customers. New customers of the bank can not be evaluated on this information.
In the cases that there was no statistical data available for the model that was developed common sense was used to make an estimation of these values. This enabled enabled us to look beyond the point of modeling and making an application based on the model.

6.10.2 Learning the model from test-data

By using old cases the model can be learned. In the current versions of the used software this feature was not available however. Nevertheless, one of the developed applications implements a way to compare the results generated by the model and those obtained by using the traditional methods. Of course it would be better if the definition values of the model could be calculated from test-data directly.

6.10.3 Expert knowledge and experience

Obviously one of the better methods to obtain definition values is to interview an expert. His domain knowledge and experience is very valuable where modeling is concerned. It is difficult for the expert to come up with exact numbers, therefore in many cases it is better to let him draw bar graphs for instance or express the relationships between the different values in another way.

Income Prospects	High		Medium		Low	
Income Debt to Ratio	Passed	Failed	Passed	Failed	Passed	Failed
Yes	0.8	0	0.6	0	0.5	0
Most Probably	0.1	0	0.2	0	0.3	0
Probably	0.05	0	0.15	0	0.1	0
Maybe	0.035	0.1	0.05	0.05	0.075	0
No	0.015	0.9	0	0.95	0.025	1

Table 6.1: Definition of node *Capital*

Think for instance about the node *Capital*. This node has five different outcomes dependent on the outcomes of the two node *Income Debt to Ratio* and *Income prospects*. Table 6.1 listst the different values for the various outcomes of these two nodes.

An expert can not come up with these numbers. It is easier to draw a bar graph like in figure 6.9 where he can draw visually the relationships between the different values. In this respect it is very important not to display the ticks and the values on the vertical axis. Drawing them would give the expert the possibility to think of a value and marking it on the graph. In cases like this however therelationships between the values is more important and turns out to be more precise when the ticks are not marked on the axis.

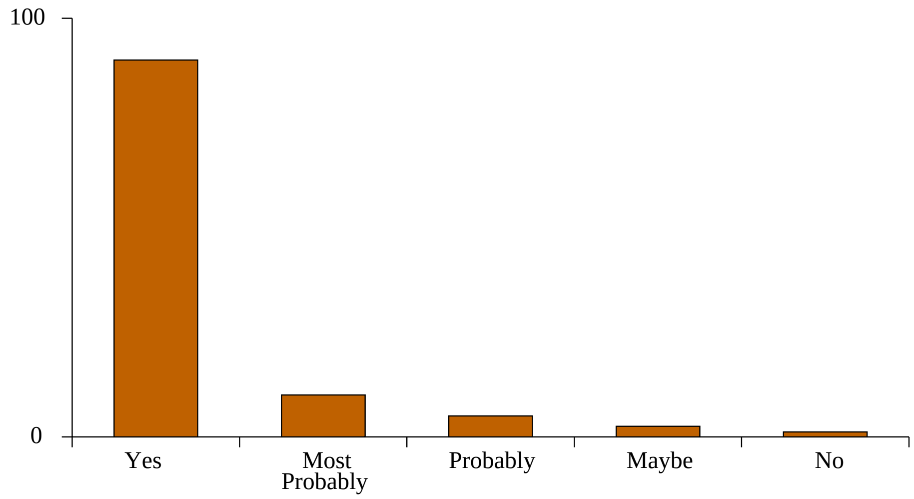


Figure 6.9: Elicitation of first column of the definitioin of node *Capacity*

6.11 Prototype

The model described in this chapter has been entered in GeNIe . In order to be able to obtain decision support information from this model we had to enter the definition values according to our best knowledge. Some further research has to be done on this subject, but the current values are quite close to the correct values. The model turns out to give the same results as the cases evaluated by human judgement in 82% of the cases. In figure 6.10 an example of the assumed definition values is given.

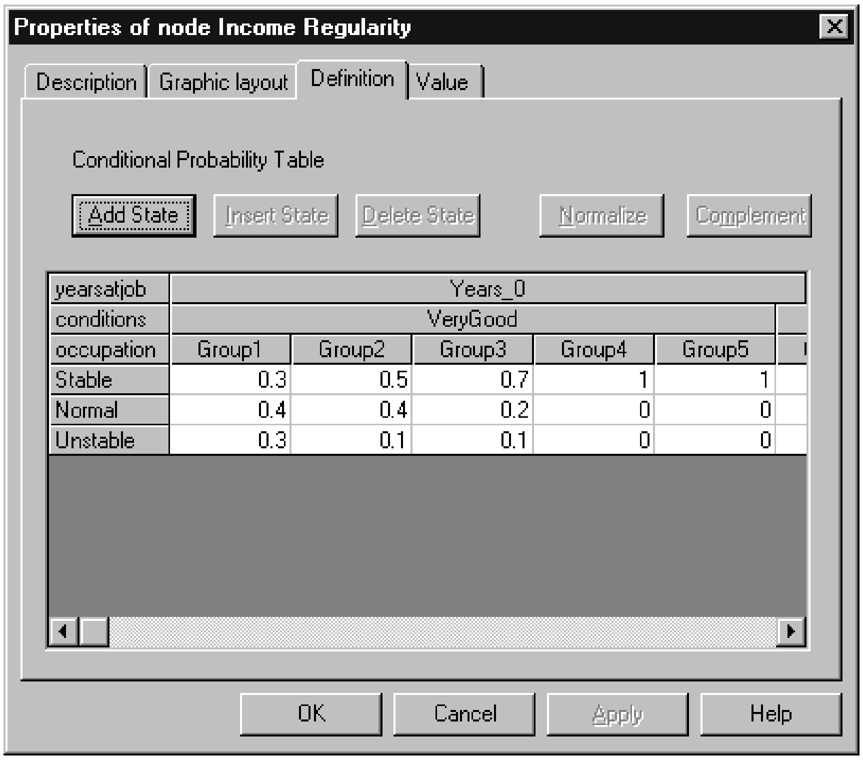


Figure 6.10: Example node definition values

Chapter 7

Applications of the Model

7.1 WWW-interface

The web-interface developed for this project is intended for use by a customer. He fills out the form and with the push of a button it calculates his chance of getting the credit he applied for.

In the privacy of his own home the customer can fill out the application without being afraid that the information he supplies will end up in the computers of the bank or in the hands of others, after all the calculations will be done on his own computer.

When the button *Calculate* at the bottom of the form is pushed, the information is fed into the Bayesian network and the result values are calculated. The systems displays the monthly loan payment and whether the credit is granted or denied.

When the loan is denied the system tries to find a way to adjust the values entered by the customer in order to generate suggestions. For instance, the system can detect if it is usefull for the customer to report more assets or that he should pay off his car and then try to apply again.

Note that some evidence can not be retrieved, like for instance the *Loan officer intuition*. In this application we do not set evidence for these nodes, instead we let the Bayesian network use the statistical probabilities of these variables, that are defined in the model.

7.2 Excel-interface

Spreadsheet programs are very widely used in the financial business. Many banks already work with models for credit evaluation implemented in one of

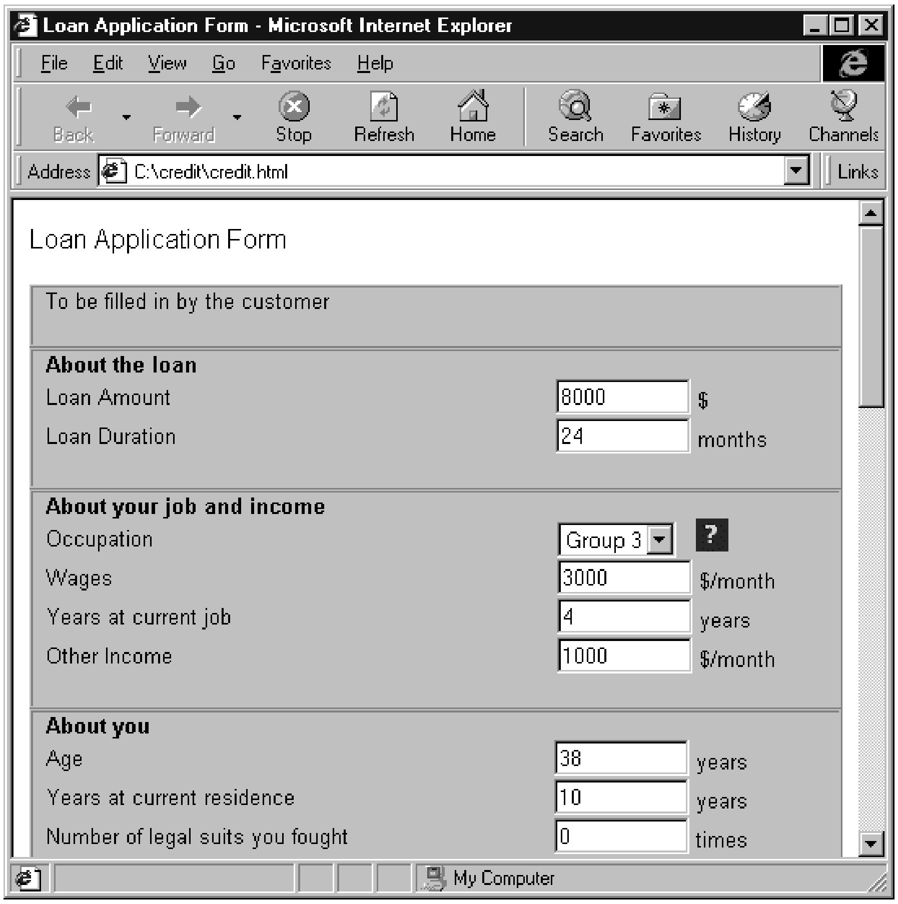


Figure 7.1: Interactive form for credit application on the Internet

these spreadsheet programs. Since these calculation toolboxes do not support Bayesian network calculations, these models are not based on the techniques used in probabilistic reasoning. By using the ActiveX control as some kind of plugin for in a spreadsheet, we have achieved the possibilities to use Bayesian models and calculations in the application.

The Excel interface is primarily intended to be used by the an employee of the bank. In particular it is intended to help credit analysts with their work. We chose the spreadsheet program Excel because it is widely used (also in banking environments) and because it supports ActiveX controls in documents.

7.2.1 Simple case evaluation

As with the WWW-interface it is possible to evaluate individual cases with the Excel Interface. One of the advantages of Excel is however, that its built-in calculation abilities can be used to generate the values that have to be fed into the model.

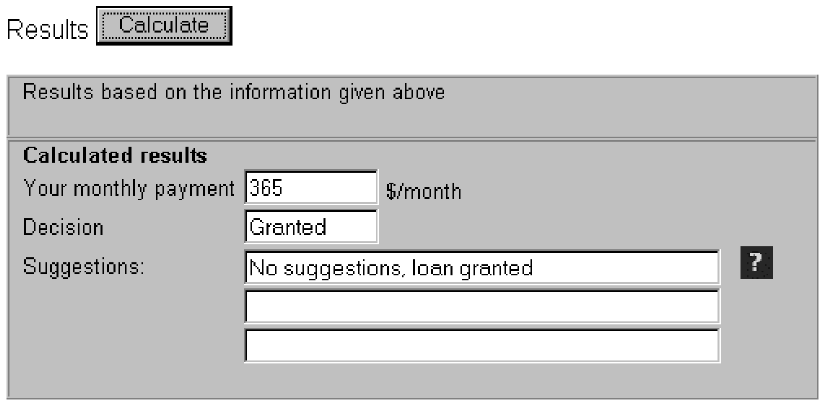


Figure 7.2: The loan is granted

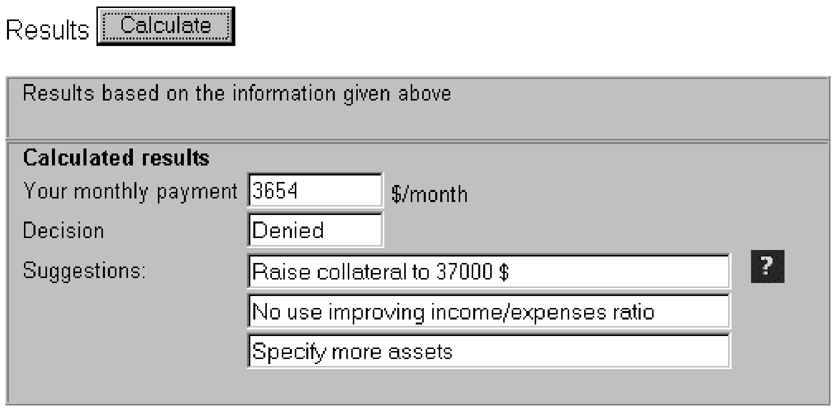


Figure 7.3: The credit is denied

As we notede already, some of the answers to the questions on the electronic application form have to be manipulated first in order to make use of them in the model. Remember for example the Debt-to-income ratio that can not be evaluated directly by the algorithms in the SmileX control.

The nescessary calculations can be done in a Visual Basic for Applications routine, like in the WWW-interface, but in Excel it is easier to have Excel figure these numbers out with the built-in mathematical abilities. This is especially usefull for people that are used to Excel anyway, so that they can easily understand what is happening. It even enables them to use their exisiting speadsheet calculations along with the new, Bayesian network based, algorithms.

The credit-application information can be entered in the form in the same way as in the WWW-interface. Since this Excel-interface is intended for the

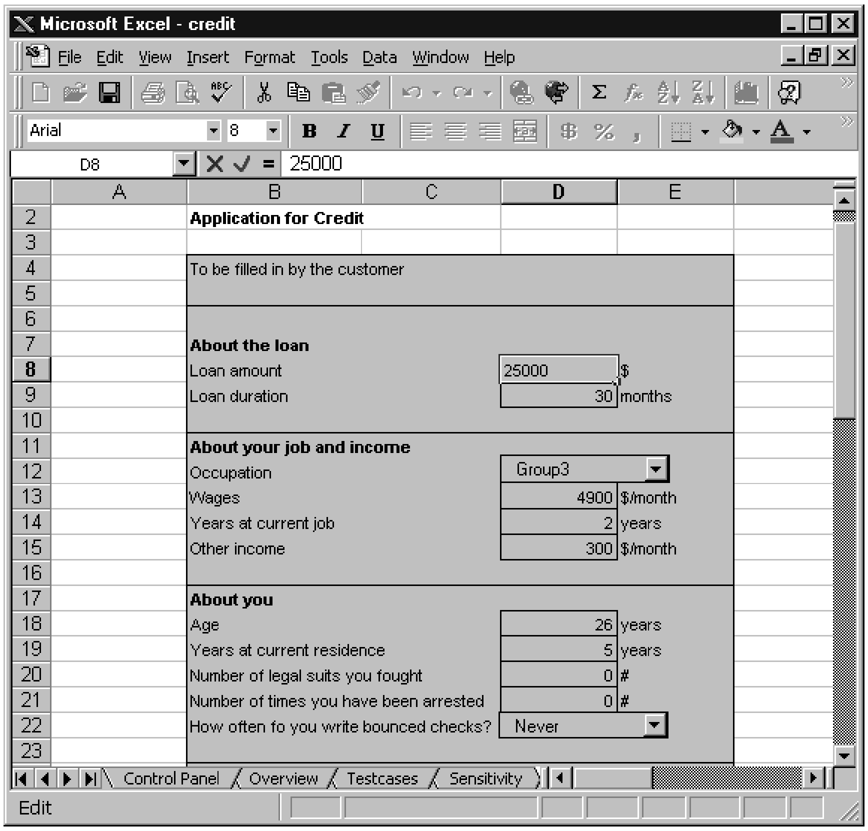


Figure 7.4: Interactive application for for the client information

bank-employees, it is also possible here to enter the evidence for the particular case that can not be determined by the client himself. See figure 7.5.

7.2.2 Sensitivity analysis

With sensitivity analysis one can see which variables in a model are prone to have direct impact on the ultimate decision that has to be taken. By altering the values of the variables one by one it is possible to see what result it has on the final decision. For instance, it is possible that a person may be denied a loan, but when he paid off his car he would be granted the same loan. In cases like this the system would look at the monthly expenses and try to vary this value in order to change the decision from *denied* to *granted*.

By altering the values of the variables one by one, one is performing a so-called one-way sensitivity analysis. Usually the results of this process are displayed in a tornado diagram.

The tornado diagram in figure 7.6 shows clearly which variables of the model

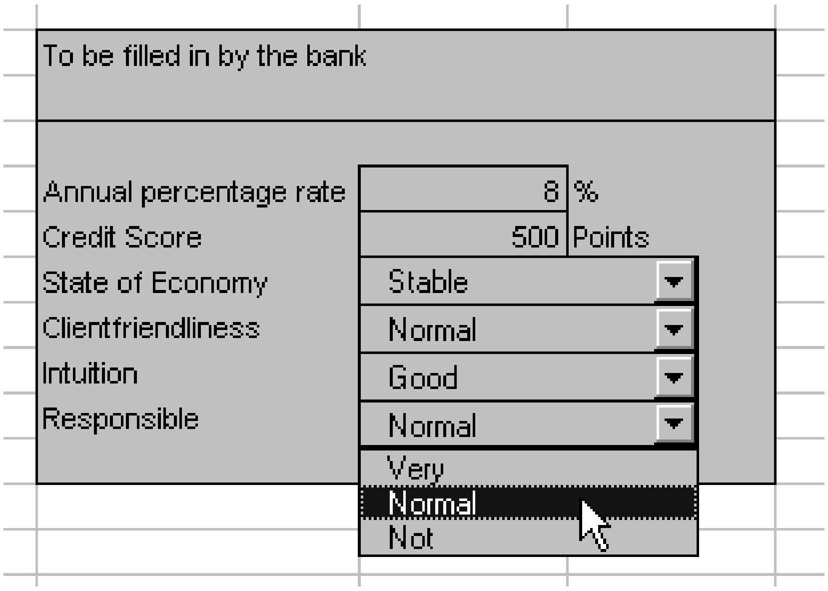


Figure 7.5: Form to be filled in by the bank

will result in immediate change of the final decision. The bars that cross the y-axis represent the possibility of changing the decision by altering the value of that variable. In order to do this one has to define for each variable a space in which the value of the variable can be varied.

The Excel interface of the model can perform a sensitivity analysis and automatically plot a tornado diagram for the model. A general graph (figure 7.6) can be plotted, using the mean values as a starting point for all the variables, but it is also possible to draw such a graph for each particular case. For each set of values (answers to questions on the application form) it is possible to see directly in the graph which of the values can be altered to change the decision.

This is particularly usefull when it comes to taking the Fair Lending Act (section 5.2)into account. According to this act it is not permitted to base the decision of granting or denying credit only on the age, sex or occupation of the applicant. Apparently it is legal to have the decision depend on a combination of these variables. In other words, when the tornado diagram does not cross the Y-axis for each of these variables, one is acting according to the law.

7.2.3 Fine-tuning the model

It is in the banks best interest to fine-tune the model to its needs. All the definition values present in the model can be adjusted to the banks preferences. To see the results of such a change it is nescesary to inspect the differences between the decisions taken with the old values and those taken with the new values. This comparison has to be made also when one wants to see how the model

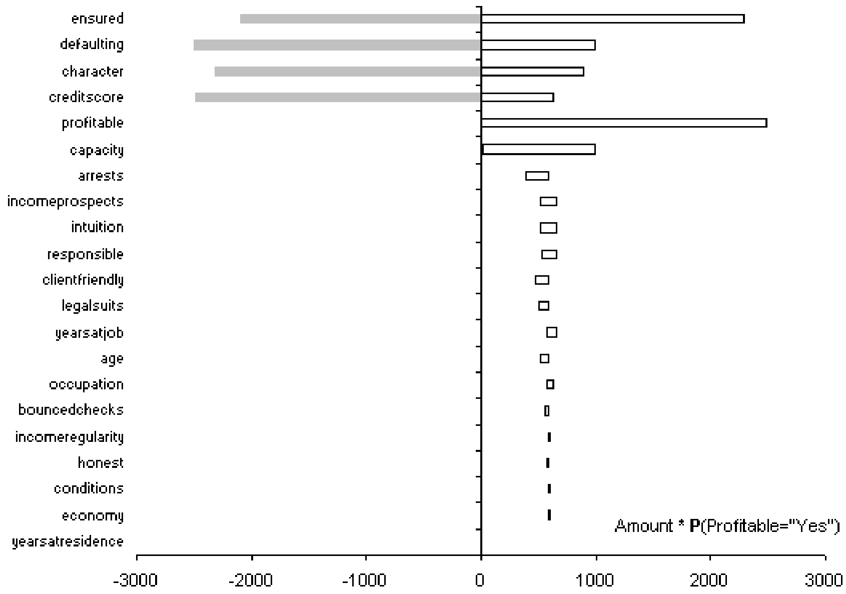
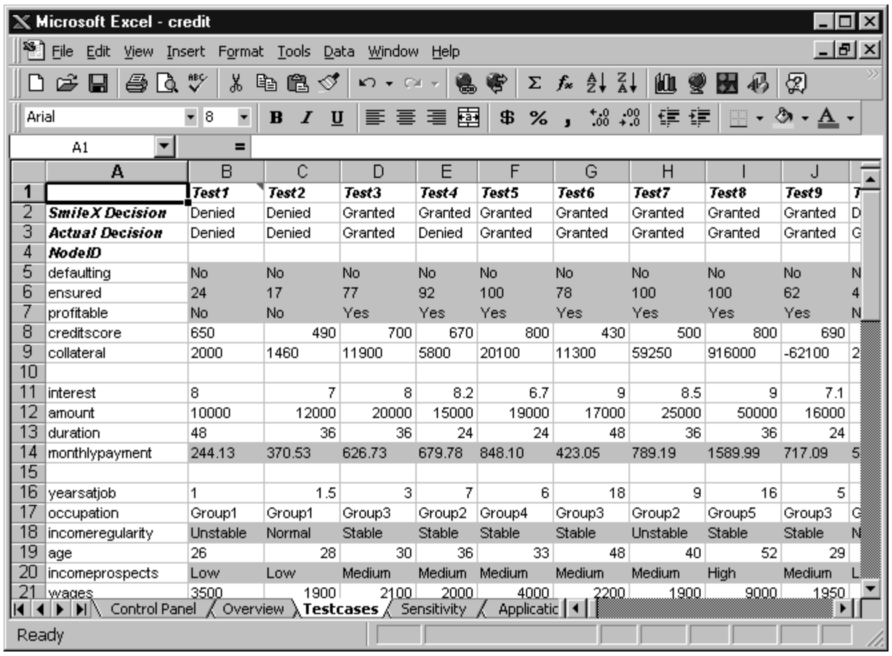


Figure 7.6: Tornado diagram generated by the Excel application

performs compared to the original (man-made) decisions.

The Excel interface makes all this possible by implementing a worksheet in which a batch of test-cases can be defined. The testcases will be run trough the model and the resulting decision will be printed at the top row of the test-case. These results will automatically be compared to the decision that is predefined. For example, if a decision based on the model is 'Granted' and the decision made by a previous version is Denied, the system will see this and alert the credit analyst that is making the changes in the model.

The credit analist can react on these alerts by canceling the changes in the model or by accepting the changes. With this worksheet it is easy to determine the impact of a change in the definition values of the model.



	A	B	C	D	E	F	G	H	I	J
1		***Test1***	***Test2***	***Test3***	***Test4***	***Test5***	***Test6***	***Test7***	***Test8***	***Test9***
2	***SmileX Decision***	Denied	Denied	Granted	Granted	Granted	Granted	Granted	Granted	Granted
3	***Actual Decision***	Denied	Denied	Granted	Denied	Granted	Granted	Granted	Granted	Granted
4	***NodeID***									
5	defaulting	No	No	No	No	No	No	No	No	No
6	ensured	24	17	77	92	100	78	100	100	62
7	profitable	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
8	creditscore	650	490	700	670	800	430	500	800	690
9	collateral	2000	1460	11900	5800	20100	11300	59250	916000	-62100
10										
11	interest	8	7	8	8.2	6.7	9	8.5	9	7.1
12	amount	10000	12000	20000	15000	19000	17000	25000	50000	16000
13	duration	48	36	36	24	24	48	36	36	24
14	monthlypayment	244.13	370.53	626.73	679.78	848.10	423.05	789.19	1589.99	717.09
15										
16	yearsatjob	1	1.5	3	7	6	18	9	16	5
17	occupation	Group1	Group1	Group3	Group2	Group4	Group3	Group2	Group5	Group3
18	incomeregularity	Unstable	Normal	Stable	Stable	Stable	Stable	Unstable	Stable	Stable
19	age	26	28	30	36	33	48	40	52	29
20	incomeprospects	Low	Low	Medium	Medium	Medium	Medium	Medium	High	Medium
21	wages	3500	1900	2100	2000	4000	2200	1900	9000	1950

Figure 7.7: Overview of testcases

Part III

Conclusion

Chapter 8

Recommendations and concluding remarks

This chapter gives some recommendations about possible extensions of the applications developed in this projct and it gives some insight in the future developments and plans with the results of this thesis.

8.1 Recommendations

8.1.1 Changes and enhancements in SmileX

The SmileX control can already be used in applications. However, in order to make it commercially attractive some features would need to be added or changed.

Optional parameters All methods implemented by the SmileX control expect a fixed set of parameters. One of the features of Visual Basic and ActiveX is that they can both be used to make methods or procedures expect only required parameters, optional parameters are used when specified by the user, but not required.

Encrypting the network For some applications it would be better if the Bayesian network or influence diagram used by the control could be encrypted in some way. It is understandable that a bank does not wish to give away the structure and definition of the network used for evaluating credit applications.

Certification This process involves submitting the code of the control to a company that can certify the code to be safe and provide a certificate of safeness. Users that browse the web and encounter a site that uses the control will then see that there is an ActiveX control on the site, but that it is safe to use it. In fact obtaining this certificate is a formality, but needed nevertheless.

JNI/Java version of SMILE To make the SMILE engine useable in a wider range of languages and platforms it could be worth trying an implementation in Java. Either the whole engine could be rewritten, or only a wrapper, using the Java Native Interface, could be developed.

ATL instead of MFC Currently the ActiveX control is implemented in C++. It makes use of a library called *MFC* (short for *Microsoft Foundation Classes*). This library offers extensive support for all sorts of functions and procedures, but, as a result, has the disadvantage that it is quite large. When the ActiveX control is used through the web, the library has to be available on the client machine. This means that it has to be downloaded (automatically) from the server whenever it is not available. When we remember that it is quite large, we can imagine that it would be better to make the control independent from this library. This can be done by using *ATL* (short for *ActiveX Template Library*). Actually, this is not a real library but rather a (minimal) framework for ActiveX controls.

8.1.2 Enhancing the creditworthiness evaluation model

Throughout the report some ways have been described already to enhance the model. They are repeated here along with some other recommendations.

Acquiring the definition values from an expert In the course of this project several experts in the credit evaluation field have commented on the structure of the model. No comments were given however on the numerical definition of the causal relations, also called definition values of the model, throughout this report. By using the elicitation techniques described in section 6.10.3 one can obtain these number and, as a result fine-tune the model according to the experience and knowledge of the expert.

Learning the model with old cases Besides acquiring the numbers from experts it would also be a good idea if the model was tested or trained by using real cases. One of the problems in this respect is that one is not sure whether an applicant, that was denied credit, would turn out to be profitable. In the next version of GeNIe there will be features for learning models with data. With these features one will also be able to detect unnoticed causal relations between variables. This technique is also called *Datamining*.

More states/choices Some of the variables can have only a limited set of values. For instance the variable occupation can have only 5 different values, depending on the type of job of the applicant. It would be better if there was a larger choice in job types or maybe even an choice based on job description. Adding more states or choices would make the model more realistic and also easier to use.

Better modeling of the purpose of the loan In the current model the purpose is modeled. Due to the fact that some numerical calculations are done in the application program code, the purpose has no real impact on the decision, but only on the loan amount.

8.1.3 Adding features in the applications

Sensitivity analysis Currently the Excel and web-applications support one-way sensitivity analysis. This is nice, but in many cases a multiple-way sensitivity analysis would be prefered. Where the current systems can only give only suggestions to clients where a change in one variable will change the resulting decision, the intended system would be able to base the suggestions on changing more than one variable at a time.

Value of information By using values of information techniques banks can minimize costs by evaluating a case before all the information is known. For instance one can evaluate a case before the credit score is obtained from the credit bureau. Depending on the case the model could be adapted in such a way that the 'need' for obtaining the credit score is calulated. When there seems to be no clear gain in obtaining this score the bank can decide to save their money and go ahead without this information.

8.2 GeNIe and SMILE

This section describes some features that the current versions of GeNIe and SMILE lack. In some cases these features are already being developed, but for reasons of completeness they are listed here anyway, since the absence of them had great impact on the project.

Continuous variables The models used in GeNIe and SMILE can not contain continuous variables. When needed they have to be discretized. For example, instead of having age groups defined one could define a continuous variable age.

Calculations In the various applications developed for this project we had to do some calculations based on values entered for variables in the model. For instance, we had to sum up all the monthly expenses like car paymanets, monthly loan payment and such. Doing these calculations directly in GeNIe would make the model less depending on an application and it would become easier to use the model.

Learning The current Excel-application can only analyse cases. It can detect when the model clearly has other results than the system with which the cases have been evaluated before. Learning the model would enable you to feed this information about differences back into the model, thereby fine-tuning it according to the results from the test-data.

Tornado diagrams Performing one-way sensitivity analysis is a good way to test the sensitivity of one variable of the network. Adding a feature that can display a tornado diagram in GeNIe would give users a clear insight in the sensitivity of the model.

Java implementation Lately Java has gained a large popularity among programmers. It could be worth to consider rewriting the whole SMILE engine in Java. Since it is now written in C++ code, it is necesarry to recompile the engine for other platforms. The Java version however would be available for every other platforms without recompiling, due to the architecture-independent nature of java.

8.3 Final Conclusion

8.3.1 SmileX

In the first part of this project an ActiveX control was developed. This development may seem a simple programming job, but it is not. In this case it involved the whole process that a commercial product goes trough:

- Problem detection and description
- Design and planning
- Implementation
- Writing documentation (see Appendix B)
- Developing example applications
- Developing a test-environment
- Developing an automated installation program
- Betatesting
- Supporting users
- Bug-fixing and adding features

The first users of the results of this thesis were students in Decision Analysis classes at the University of Pittsburgh and Carnegie Mellon University. These students had to design their own decision problems and build an application around this problem based on the SmileX ActiveX control.

In the near future the package will be made available for everyone. It will, like all software developed at the DSL, be downloadable from the web and free for personal use.

8.3.2 Evaluating creditworthiness

The second part of the project has resulted in a large amount of research. A model has been built that represents the decision problem of a bank trying to evaluate a customer, who is applying for credit. Besides giving the bank the opportunity to get decision support we supply the customer with an interactive form that predicts his chance on being approved for the credit.

Another important feature of using Bayesian network technology for this problem is that one can generate suggestions. A customer that will be denied credit based on the given information is given suggestions about the way to improve his application.

Bayesian network models and influence diagrams are a good and clear way to gain insight in decision problems and processes. Besides the fact that it is a good excercise to develop these networks they are an excellent way to communicate ideas about decisions.

Of course banks are not so anxious to give away their view on evaluating consumer creditworthiness. Goverments or controlling institutes could however consider to force banks that use this method to reveal (to them) the model in order to be able to check the correctness and whether it is legal practice.

By using this model in a tool in which we can easily check sensitivity for certain variables and evaluate the performance of the model against the traditional system, we can fine-tune the model and improve the resulting predictions. This tool, intended for usage in a bank, has also been developed in this project.

Even though the model was designed to be useable for banks in the United States of America, a company in the Netherlands has already shown its interest in the model and will soon start a project that uses this model as its basis.

Part IV
Appendices

Appendix A

Definition and term-list

Assets Property and possessions regarded as having value in meeting debts or commitments.

Credit Transaction between two parties in which one (the creditor or lender) supplies money, goods, services, or securities in return for a promised future payment by the other(the debtor or borrower). Such transactions normally include the payment of interest to the lender. Credit may be extended by public or private institutions to finance business activities, agricultural operations, consumer expenditures, or government projects.

Credit Bureau Organization that provides information to merchants or other businesses relating to the credit worthiness of their customers. Credit bureaus may be private enterprises or operated on a cooperative basis by the merchants in one locality. Users of the service pay either a fee based on the amount of service or a flat membership charge.

Credit Report Historical financial and employment information relating to the loan applicant, maintained by credit bureaus.

Consumer Credit Short- and intermediate-term loans used to finance the purchase of commodities or services for personal consumption, or to refinance debts incurred for such purposes. The loans may be supplied by lenders in the form of cash loans or by sellers in the form of sales credit.

Collateral Security pledged as a guarantee for repayment of a loan. It may be in the form of property, stocks and bonds, cash, or other valuables.

Consumer A purchaser of goods or services. For our purposes, we will consider a consumer a customer of the bank who wishes to be granted a loan.

Disposable Income That portion of an individual's income over which the recipient has complete discretion. An accurate general definition of income is not easy to provide. Income includes wages and salaries, interest and dividend payments from financial assets, and rents and net profits from other businesses. Capital gains on real or financial assets should also be counted as income in most cases, at least insofar as they increase spending power. Such gains may even be counted where the asset is not actually sold, and the increase in spending power is not exercised.

Lender A party who agrees to give the recipient a sum of money on the condition that it will be repaid with interest over an agreed upon period of time.

Loan A sum of money given to the recipient on the condition that it will be repaid with interest over an agreed upon period of time.

Margin In finance, the amount by which the value of collateral provided as security for a loan exceeds the amount of the loan. This excess represents the borrowers equity contribution in a transaction that is partly financed by borrowed funds; thus it provides a 'margin' of safety to the lender over and above the collateral that is pledged.

Repossession Regaining possession of property or goods in which repayment of a debt is in arrears.

Security on loans In the event of a borrower's bankruptcy, the lender may have to share the borrower's assets with competing creditors and may receive only partial satisfaction or even none at all. Lenders therefore, urge borrowers to give security for the loan.

Appendix B

SmileX documentation

B.1 Introduction

B.1.1 About this manual

This is the manual for the SmileX ActiveX control. This is **NOT** the HTML-version of the manual for the Smile C++ library! The C++ documentation is currently only available in MS Word (`.DOC`) format. You need this manual when you create an application around a belief network.

This manual describes the Methods and Properties of the SmileX ActiveX control. It contains a complete reference to all the supported features and an extensive amount of examples. A lot of languages can be used to interact with ActiveX controls the largest part of this manual will will focus however on examples in Visual Basic.

B.1.2 What is SmileX?

Imagine a bank that has created a belief network to support the decision whether or not to grant a loan to a customer. Instead of using GeNIe to determine the outcome of the decision support network the bank wants to create it's own program/user interface based on the technology used in GeNIe. And most of all, it wants to publish a web-page where customers can find out if they are eligible for a loan. SmileX makes this possible!

SmileX is an ActiveX control based on Smile. It offers you the possibilities to write an application using Smile-technology in a wide variety of languages. Instead of being tied to C++ (using the Smile C++ classes) you can now use Visual Basic, J++, Delphi or any other environment that supports ActiveX controls. You can for instance use the control in Web-pages or in Excel, using VBScript or VBA respectively to access the methods and properties of the control.

SmileX is limited however. It offers a subset of the functions and procedures in Smile. This subset includes all the functions you need to extract decision support information out of a network, created using GeNIe or Smile. In other words: the ActiveX control can not create or modify the definition of a network, but it can give you decision support results based on specific information you offer.

B.1.3 Using SmileX

Everyone who wants to use decision support in their applications can use the SmileX ActiveX control. All you need is an existing network model for the decision you want to make, an application environment that supports ActiveX (A web-browser, C++, Visual Basic, Excel etc...) and of course the SmileX control.

Using SmileX for personal use is free! If you plan to use it for commercial purposes however, you should contact us.

B.1.4 Questions and Remarks

If you have questions or remarks about SmileX, we will be happy to hear from you. For general questions about Bayesian Networks, Influence Diagrams etc. we direct you to the online-help of GeNIe

B.1.5 About the examples

The examples in this manual all depend on one of two network (.DSL) files, animals.dsl and pinball.dsl. These files should be available in the directory you installed the SmileX development package in. The installation procedure will update the paths to these files automatically.

B.2 Properties and Methods

This chapter gives a brief description of the features currently implemented in the SmileX control. Examples of the usage of these functions will be ginven in the next chapter.

B.2.1 Properties

NetworkID

This is a read-only property. It is simply the ID of the network given by the developer/designer of the network.
Example:

```
MyNetworkID = SmileX.NetworkID
```

This example declares a variable MyNetworkID and assigns it the ID of the currently loaded network.

NetworkName

This is also a read-only property. It is the name of the network as provided by the network developer/designer.

```
MyTextBox.Value = SmileX.NetworkName
```

This example assigns the name of the currently loaded network to the value property of the MyTextBox object.

B.2.2 General network methods

ReadFile(filepath)

This method reads a network file (.DSL) and creates the appropriate structure in memory. The filepath parameter can contain the whole path of the file. On success this method returns 0, on failure it returns a negative value. Only one network can be loaded into the ActiveX Control instance. Insert more instances if needed.

```
SmileX.ReadFile("c:\smilex\asia.dsl")
```

This reads the file Asia.dsl in the directory c:{}smilex{}asia.dsl.

ReadWebFile(httpadres)

The ReadWebFile method can read a network file (.DSL) on a remote adres. It actually downloads the file to the local Internet cache and reads it. The httpadres parameter holds the complete adress of the file (including including the filename itself e.g. 'http://www.sis.pitt.edu/ dsl/asia.dsl' The return value of this method is 0 on success, and a negative value on failure. Again only one network can be loaded into one ActiveX Control Instance

```
Result = SmileX.ReadWebFile( _
         "http://www.sis.pitt.edu/~dsl/animals.dsl")
If Result = 0 Then
  MsgBox "Successfully read the file"
Else
  MsgBox "Failed to read the file"
End If
```

This loads the file 'animals.dsl' into the control, provided that the file is published at the location in the httpadres parameter.

GetAllNodes

This method returns an array (a variant containing an array) of strings. As you might have guessed from the name these strings represent the id's of all the nodes in the network.

```
TheNodes = SmileX.GetAllNodes
If Not IsEmpty(TheNodes) Then
  For Each Node In TheNodes
    MsgBox Node
  Next
End If
```

This first line in the example will assign the variable TheNodes an array of string values being the ID's of the nodes. You can use these ID's in calls to a lot of other methods described in this document. The If-statement in the example checks if the resulting array is empty as a result of either an error or of the fact that the loaded network has no nodes. If the resulting array is not empty in the example a message-box is displayed on the screen for each node-id in the list.

B.2.3 General node methods

GetOutcomes(nodeid)

The GetOutcomes method returns an array of strings, being the outcomes of the specified node. Make sure you provide the node-id and not the nodename or comment. On failure this method returns an empty variant.

```
TheOutcomes = SmileX.GetOutcomes(MyCombobox.Value)
```

This example assigns an array of the outcomes of the node, which id is specified in the combobox MyComboBox. If the node has no outcome (e.g. if it is a Value node) or if the node secified does not exist in the loaded network, the variable TheOutcomes is empty (which can be detected by the IsEmpty function as described in the GetAllNodes method)

GetIndexingParents(nodeid)

This method returns an array of strings, being the nodeid's of the nodes that are indexing parents to the node specified in the nodeid parameter. On failure an empty variant is returned.
Important: Indexing parents are not not nescesarily the nodes that are the direct parents in the model. They are rather the nodes that index the value of the node. The different combinations of the outcomes of these indexing parents show up in GeNIe on the value tab of a node.

```
TheIndexingParents = SmileX.GetIndexingParents("Node0")
```

The variable TheIndexingParents is assigned an array with the indexing parents of the node with id 'Node0'. If the node does not have any parents or does not exist, an empty value is returned.

GetNodeName(nodeid)

This method returns the name of the node of which the nodeid is specified in the nodeid parameter.

```
TheName = SmileX.GetNodeName("Node0")
If Not IsEmpty(TheName) Then
  MsgBox TheName
Else
  MsgBox "Could not find specified node"
End If
```

The variable TheName is assigned the result of the GetNodeName function this result is empty if the specified node is not found. It contains the name of the node when the node is found.

GetNodeComment(nodeid)

This method returns the comment of the node of which the nodeid is specified in the nodeid parameter.

```
TheName = SmileX.GetNodeComment("Node0")
If Not IsEmpty(TheComment) Then
  MsgBox TheComment
Else
  MsgBox "Could not find specified node"
End If
```

The variable TheName is assigned the result of the GetNodeComment function this result is empty if the specified node is not found. It contains the comment of the node when the node is found.

B.2.4 Evidence methods

SetEvidence(nodeid, outcome)

SetEvidence sets the value, specified in the outcome parameter as evidence for the node specified in the nodeid parameter. When this operation was successfull 0 is returned, a negative value otherwise.

```
Result = SetEvidence("Class", MyComboBox.Value)
If Result
...
```

We set the state specified in the combobox MyComboBox as the evidence for node 'Class' here. If that failes the message 'Failed!' will be shown on the screen. On Success the message 'Success!' will show.

GetEvidence(nodeid)

This method returns the evidence of a node(the one specified in the nodeid parameter) that was set using the Setevidence method.

```
MsgBox SmileX.GetEvidence("Class")
```

The messagebox in this example shows the name of the state or choice set as evidence for this node. If there is no evidence set for this node the messagebox will be empty.

ClearEvidence(nodeid)

ClearEvidence retracts the evidence set for a node. It does not test whether evidence for the node was provided in an earlier stage, it rather clears the evidence blindly. If nothing went wrong 0 is returned, a negative value otherwise.

```
Result = ClearEvidence(MyComboBox.Value)
```

After execution of this statement the value result holds a 0 if the evidence of the node ,specified (selected) in the combobox MyComboBox, is cleared successfully. If not successfull (e.g. node with specified ID does not exist) Result contains a negative number.

ClearAllEvidence

This method clears any evidence that was set for any node in the network.On Success it returns 0, on failure a negative value is returned. This method does not require any parameters.
Example:

```
Result = SmileX.ClearAllEvidence
```

The variable result will contain a 0 if the evidence for all nodes in the currently loaded network is cleared successfully. A negative number otherwise.

B.2.5 Target Methods

SetTarget(nodeid)

The SetTarget method sets the Target flag of the node specified in the nodeid parameter. This means that its values will be regenerated when UpdateBeliefs is called. On success this method returns 0, a negative value otherwise.

```
Result = SmileX.SetTarget(MyTextBox.Value)
```

The node specified in the textbox MyTextbox will be used as a target next time we use the UpdateBeliefs method, that is, if Result holds a zero (0) after executing this statement, if it holds a negative number the method failed to set the specified node as a target.

IsTarget(nodeid)

This method returns a boolean value. True when the node specified has its target-flag set. False otherwise.

```
Result = SmileX.IsTarget("Node0")
If Result = True Then
  MsgBox "Node0 is a target"
Else
  MsgBox "Node0 is no target"
End If
```

This example determines wheter 'node0' is set as a target in the currently loaded network. If so (the value of Result is True) then the message 'Node0 is a target' is displayed. If not, the corresponding message is shown.

ClearTarget(nodeid)

This method clears (unsets) the target flag of the node specified in the nodeid parameter. On success a 0 is returned. On failure a negative value.
Example:

```
Result = SmileX.ClearTarget("Node0")
```

This example assigns a result value to the variable result, depending on the succcess or failure of the method. On success ('Node0' is no longer a target node) Result will be 0. On failure (e.g. 'Node0' does not exist) Result will contain a negative value.

B.2.6 Belief Methods

UpdateBeliefs(Algorithm)

This method updates the beliefs in the network. The parameter algorithm should a string , specifying to the update algorithm to be used. On Success this method returns 0, a negative value otherwise. If you specify an invalid string this method will use the default algorithm and update the network successfully, but return a negative value.

String-Value	*Update algorithm for*
Lauritzen	Bayesian Networks
Henrion	Bayesian Networks
Pearl	Bayesian Networks
LSampling	Bayesian Networks
SelfImportance	Bayesian Networks
HeuristicImportance	Bayesian Networks
Backsampling	Bayesian Networks
CooperSolving	Influence Diagrams

Table B.1: Updating algorithms

```
Result = SmileX.UpdateBeliefs("Lauritzen")
```

This updates the currently loaded Bayesian Network according to the Lauritzen algorithm. If the currently loaded network is an Inference Diagram then the default algorithm set in the loaded network will be used and a negative integer will be returned and Assigned to the variable Result.

GetBeliefs(nodeid)

This method returns an array of doubles, being the beliefs for the different outcomes of the node (a.k.a. the beliefvector). To find out which belief corresponds with which outcome, use the GetOutcomes method. The corresponding values have the same index in the resulting arrays.

```
TheBeliefs = SmileX.GetBeliefs("Node0")
If Not Isempty(TheBeliefs) Then
  For Each Belief In TheBeliefs
    MsgBox Belief
  Next
End If
```

This example puts the beliefvector (an array of doubles) in the variable TheBeliefs. On success (i.e. the array is not empty) the elements in the vector are displayed in a messagebox one by one.

GetBeliefForOutcome

This method returns the belief (a double) for the outcome specified in the outcome parameter of the node corresponding to the nodeid parameter. On failure this method returns an empty Variant.

```
TheBelief = SmileX.GetBeliefForOutcome("Class","Bird")
```

The variable TheOutcome will contain the belief (probability) of the outcome 'Bird' of the node node 'Class'.

GetExpectedUtiliy(nodeid, outcomes)

GetExpectedUtility returns the expected utility of a node, depending on the specified outcomes of its (indexing) parents. Specifying these outcomes is equivalent to finding the right column on the value tab of a node in GeNIe. On success this method returns the proper expected utility (a double) on failure it returns an empty variant. If you do not want to specify the outcomes of the parents, use the GetBeliefs method and pick the appropriate value from the resulting array (use the trial-and-error algorithm :-)

```
TheOutcomes = ("","")
ReDim Preserve TheOutcomes(4)
For Index = 0 To 3
  TheOutcomes(Index) = MyListBox.List(Index,0)
Next
TheUtility = SmileX.GetExpectedUtility("Node0", _
                                        TheOutcomes)
MsgBox TheUtility
```

This example is quite complicated. This is because it has an array as a parameter. This array has to be initialised as an array of strings (hence the first line). The it has to be be dimensioned correctly, which is done in the second line (this example assumed that the node 'Node0' has 4 (four) indexing parents. The array TheOutocomes specifies the outcomes of these indexing parents, and is filled with values that have been preset in the listbox MyListbox. The nodeid and the array of the oucomes of the indexing parents are passed to the GetExpectedUtility method and the resulting double is stored in the variable TheUtility. This value is displayed in the last line in a MessageBox.

GetExpectedUtiliy2(nodeid, outcomes)

This method does the same as the GetExpectedUtility method. The only difference is that the outcomes of the indexing parents are not specified in an array, but in a comma-separated string. This enables JavaScript users, that can not create Visual Basic Arrays, to get expected utility values from the models. Of course VB programmers can also use it. Again an empty value is returned on failure.

```
result = SmileX.GetExpectedUtility2("Node0", _
```

B.2.7 User Property Methods

GetUserProperty(nodeid, propertyname)

This method gets the value of a user property. Every node can have a set of user-properties, and the network can also have a general set of of these properties. The first parameter is either the ID of the node of which you want to get the property or the literal string "[NETWORK]" (uppercase, without quotes) to get a general user property. The second parameter is the name of the property of which you want the value.

Important: Property names are case-sensitive. This means that for example one node can have a property "MyProperty" and another property "MYProperty". Obviously, these two properties can have different values.

```
result1 = SmileX.GetUserProperty("Node0","MyProperty")
result2 = SmileX.GetUserProperty("[NETWORK]","MYProperty")
```

SetUserProperty(nodeid, propertyname, value)

This method sets a user-property to the specified value. Again one can use "[NETWORK]" as the first parameter, instead of a nodeid in order to gain access to the general properties of the network. If the specified property already exists for the node (or network), the value is changed to the new value as specified in the third parameter. If there is no such property for the specified node (or the network), a new property will be created and assigned the value in the third parameter.

```
result1 = SmileX.SetUserProperty("Node0", _
                                 "MyProperty1","Foo")
result2 = SmileX.SetUserProperty("Node0", _
                                 "MyProperty2","Bar")
result3 = SmileX.SetUserProperty("Node0", _
                                 "MyProperty1","FooBar")
```

Given that node 'Node0' does not have any properties yet, before execution of this code, this example creates 2 new properties: "MyProperty1" and "MyProperty2". This is done in the first 2 statements. Respectively these two properties get the values "Foo" and "Bar". The third statement changes the value of "MyProperty1" into "FooBar".

GetUserPropertyNames(nodeid)

This method returns a list of names of properties, that are currently defined for the, in the first parameter specified, node. The general properties of the network can again be obtained by passing "[NETWORK]" as the parameter. If the specified node or the network does not have any properties, an empty value is returned.

```
result1 = SmileX.GetUserPropertyNames("Node0")
result2 = SmileX.GetUserPropertyNames("[NETWORK]")
```

After executing the first statement, result1 will contain an array of names of the names of properties that are currently known for node "Node0". If Node0 does not have any known properties, result1 will be empty. Executing the second statement will assign an array with the names of the general network properties to the variable result2.

DeleteUserProperty(nodeid, propertyname)

This method deletes a user defined property of the name specified in the second parameter from the list of the node specified in the first parameter. On succes 0 is returned, on error an empty value is returned.

```
result1 = SmileX.DeleteUserProperty("Node0","MyProperty")
```

After executing this statement, result1 will contain either a negative number (if there was an error) or 0 (on success). If result1 equals zero then the property “MyProperty” is deleted from the list of properties of node “Node0”.

B.3 Using SmileX with VB Script

This chapter describes an example of a web-based interface. All examples in this chapter will include the raw HTML-code, instead of instructions on which menu's to choose, and buttons to click in an endless number of applications. Communicating with the control involves some programming in either VB-Script or JavaScript.

B.3.1 Inserting the SmileX control

To insert the control in a web-page use the following syntax:

```
<OBJECT
   CLASSID="CLSID:035C8025-AC03-11D2-B144-0000C0E59BCA"
   CODEBASE="http://www.whatever.edu/ActiveX/SmileX.ocx"
   ID=SmileX
   HEIGHT=0
   WIDTH=0>
 </OBJECT>
```

This inserts an ActiveX object into the page with a somewhat mysteriously looking CLASSID tag. This tag is used to determine which object is to be insterted in the page and where it can be found. The registry contains a reference to the file in which the control resides. If this reference does not exsist it may mean 3 things:

1. There is no ActiveX control with that CLASSID on your machine
2. It is there, but it is not yet registered.
   You can register the control by running `regsvr32.exe` with the filename in which the control resides as a parameter. So, for example:

   ```
   regsvr32 c:\smilex\smilex.ocx
   ```

   The executable `regsvr32.exe` is usually in your windows or system directory.
3. Since the ActiveX control may be on a web-page, the control may need to be downloaded first (Downloading and registering should be handled correctly by the web-browser, this requires the security setting not to be such that they automatically reject ActiveX-objects).

The CODEBASE tag contains the web-adres where the control can be found when it is not yet available/installed/registered on the computer. You may leave out this tag if you know that you already have the control (which is obvious if you installed the package correctly). If you want to make the control available from a web-page you must include it however. (Make sure you put the control on the adress specified in the CODEBASE tag) The ID tag gives the control a name. This name will be used in the script to communicate with the control. In the examples this name will be SmileX. The HEIGHT and WIDTH tags determine the size of the object on the screen. Since it is not nescessary to view the control (the control does not even have a display method...) these tags may be set to 0.

B.3.2 General issues on accessing the control through scripting

The Script Block

On every web-page where you include the control, you will need to insert a script-block as well. The open and close tags look like this:

```
<SCRIPT LANGUAGE="VBSCRIPT">

</SCRIPT>
```

Please note that every script statement in the script block will be executed everytime after you load (or reload) the page. Only statements in subroutines will not be executed atuomatically (unless these subroutines get invoked by the code that is run automatically of course). It is also possible that (on loading) some events are generated that some subroutines are supposed to catch. These subroutines will also be executed.

Accessing properties

A property can be accessed as follows:

```
NetID = SmileX.NetworkID
```

This will assign the network ID to the variable NetID, assuming that a network-file has been loaded into the control.

Using Methods of the control

Methods can be invoked as follows:

```
Result = SmileX.ReadFile("c:\smilex\animals.dsl")
```

This loads the file $c:\backslash smilex\backslash animals.dsl$ into the control. And assigns the resulting value to the variable Result. If a method requires more than one parameter, separate the parameters by a comma:

```
Result = SmileX.SetEvidence(Node0,Choice1)
```

B.3.3 Using HTML INPUT values

This section describes some usefull HTML controls that you can use in pages where you use the SmileX control.

Textbox

```
<INPUT TYPE=TEXT NAME=MyTextBox VALUE="">
```

This adds a textbox to the HTML-page which can be referred to as MyTextBox. It's initial value will be the string " (without the quotes). Use the Value property of the textbox to pass the current value to a function or to change it.

```
MyTextBox.Value = SmileX.GetEvidence("MyNode0")
```

This bit of code finds the status or choice that is set as evidence for the node MyNode0 and displays it in the textbox MyTextBox.

```
SmileX.ReadFile(MyTextBox.Value)
```

This statement reads a file into the control. The parameter filename is retrieved from the textbox.

Button

A button is often used in HTML-pages to perform a certain action. Sometimes it will submit values and bring you to another page, but this is not obligatory. You can also use a button to invoke functions of the SmileX control. A button is inserted into a HTML page as follows:

```
<INPUT TYPE=BUTTON VALUE="OK" NAME="MyButton">
```

This will insert a button into your page with the caption 'OK' (no quotes) and the name MyButton. The name is crucial when catching the events fired by the control. The event you will propably need to catch is the OnClick event. Catching this event can be done by entering a VBScript subroutine in the VB Script block:

```
Sub MyButton_OnCLick
  SmileX.UpdateBeliefs("Lauritzen")
End Sub
```

This subroutine causes the beliefs in the network loaded into the control to be updated using a certain algorithm. By using the _OnClick notation this action is tied to the OnClick event of the button MyButton. Remember that this not mean that you go to another web-page!

Radio buttons

When the user can make one choice out of a small number of options it may be usefull to use radio buttons. These buttons can work as a group and enabling one of them will automatically disable the rest of the buttons in the group. An example:

```
<INPUT TYPE=RADIO NAME=SEX VALUE="Male"> Male<BR>
<INPUT TYPE=RADIO NAME=SEX VALUE="Female"> Female<BR>
```

This inserts a group of two radion buttons (the buttons work as a group because they have the same NAME property) of which the user can choose one. Using the value of the group can be done as follows:

```
SetEvidence("MyNode",Sex.Value)
```

This sets the evidence specified by the choice in the radio button group for the node MyNode. Whenever the user changes the value of the radiogroup (selects another button than the one currently selected), the OnChange is automatically fired. This event can be caught by useing the following construction:

```
Sub Sex_OnChange
  SetEvidence("MyNode",Sex.Value)
End Sub
```

This sets the proper evidence whenever the user changes the selection.

Checkbox

A checkbox can be used when you want the user to be able to choose between a True or False value. You can for example let the user decide whether or not to use a certain node as a target-node:

```
<INPUT TYPE=CHECKBOX NAME=MyCheckBox
       VALUE="Use Mynode as target">
```

This creates a checkbox with the caption 'Use Mynode as target'. This chaeckbox can be referred by its name MyCheckBox. To use the status of to determine whether or not to use the node as a target requires a statement similar to the following:

```
If MyCheckBox.Selected Then
  SetTarget("MyNode")
Else
  ClearTarget("MyNode")
End If
```

Executing this statement whenever the status of the checkbox is changed can be done as follows:

```
Sub MyCheckBox_OnChange
  If MyCheckBox.Selected Then
    SetTarget("MyNode")
  Else
    ClearTarget("MyNode")
  End If
End Sub
```

Combobox

A combobox is often used when the user can make a choice out of a limited number of options. One will often find a combobox in pages where the country of residence has to be supplied by the user, the combobox then contains a list of (specified) countries. Inserting a combobox into a HTML page can be done as follows: (notice that this is not really an INPUT object, but a SELECT object!)

```
<SELECT NAME=MyCombobox>
      <OPTION VALUE="Mammal" SELECTED>A Mammal
      <OPTION VALUE="Reptile">A Reptile
      <OPTION VALUE="Bird">A Bird
      </SELECT>
```

This inserts a combobox with tree choices into the page. The choices need to be specified by a value and a name. The value of the combobox object equals the currently selected value. The initial value can be set by using the SELECTED tag in an OPTION VALUE declaration. Every option is followed by the string that is actually displayed in the combobox on the screen. In this case the option value Mammal displays the words 'A Mammal'. Using the value-propertie can be done in the same general way as the value-property of a textbox:

```
SmileX.SetEvidence("AnimalClass",MyCombobox.Value)
```

This will introduce the class chosen in the combobox as evidence for the node AnimalClass. It may be usefull sometimes to catch the OnClick event of the SELECT object. This will enable you to perform an action whenever the user click (changes) the value in the combobox. An Example:

```
Sub MyCombobox_OnClick
  SmileX.SetEvidence("AnimalClass",MyCombobox.Value)
End Sub
```

B.4 Using SmileX with JavaScript

This chapter is intended for those people that want to do the scripting in their web-pages in JavaScript. In some cases this can have advantages but in general it is equivalent to using VBScript. All HTML-commands remain the same. Most of the text in this chapter resembles the tekst in the chapter about using VB Script, JavaScript users will find some usefull additional information in this chapter however. There are two full-featured example pages written with JavaScript. The first one contains all the discussed methods and properties and can be used as a test-page for your models. The second one shows a possible implementation for an interface featuring a network that models the probability of lungcancer and tuberculosis, when going to Asia.

B.4.1 General remarks about JavaScript

The Script Tag

The script tag for Javascript code is (obviously) different from the tag of VB script code. To make sure that older browsers will not (that do not support JavaScript) parse the code you can (sort of) hide it from these browser by using the HTML comment tags. Use:

```
<SCRIPT LANGUAGE=JavaScript> <!--
 .
 alert("Hello World");
 .
// -->
</SCRIPT>
```

For reasons of simplicity this hiding of code is often not used, it is recommended however.
As with VBScript, JavaScript code in these tags will be executed everytime the page is (re-) loaded. JavaScript functions are similar to VBScript subroutines, code in these functions will not be automatically executed on loading the page. Code in functions will only be executed when such a function is called explicitly.

HTML Controls

Due to JavaScript's object-oriented nature all HTML objects (such as a textbox, a checkbox or a button) have to be referred to by their full name including the prefix “document.” and the name of the form they are in. For Example:

```
<FORM NAME=ThisForm>
  <INPUT TYPE=TEXT NAME=ThisText>
</FORM>
.
.
<SCRIPT LANGUAGE=JavaScript>
  document.ThisForm.ThisText.value = "Hello World"
</SCRIPT>
```

Path Names

Strings in JavaScript can (like in Java and C(++)) contain escape characters. These characters are usually prefixed by a backslash ('\'). A literal backslash can be constructed by using two (2) backslashes. This is particularaly important when you have to specify a path name. For Example:

```
result = SmileX.ReadFile("C:\\Examples\\network.dsl")
```

End-of-line

An end-of-line can be constructed by adding two escape characters to a string: '\r' and '\n' . For Example:

```
<FORM NAME=theform>
  <TEXTAREA NAME=thetextarea>
  </TEXTAREA>
</FORM>
.
.
<SCRIPT LANGUAGE=JavaScript>
  var themessage = ""
  for(i=0;i<10;i++)
  {
    alert("Hello World\r\n") ;
  }
  document.theform.thetextarea.value = themessage;
</SCRIPT>
```

Case Sensitive

The JavaScript syntax is similar to that of C and Java. Bear in mind that JavaScript, unlike VBScript, is case-sensitive. Referring to objects in web-pages (like e.g. HTML controls) is also case-sensitive. For Example:

```
<INPUT TYPE=TEXT NAME=CaSeSeNsItIvE>
.
.
<SCRIPT LAGUAGE=JavaScript>
  document.form.CaSeSeNsItIvE.value = "FooBar"
</SCRIPT>
```

Empty variants

Using variants in JavaScript is somewhat awkward. A variant that is empty will be equal to null in JavaScript. Since SmileX often returns empty values on error, this is quite important. For example:

```
var TheVariant = SmileX.GetAllNodes();
if(TheVariant==null)
{
  alert("TheVariant is empty") ;
}
```

Arrays

Arrays in JavaScript can only have one dimension (as opposed to multi- dimensional arrays in VBScript). Once a SmileX method has returned a VB Array it has to be 'reformatted' as a JavaScript Array. For Example:

```
var TheVBArray = SmileX.GetAllNodes();
if(TheVBArray!=null)
{
   TheJavaScriptArray = TheVBArray.toArray();
   for(i=0;i<TheJavaScriptArray.length;i++)
   {
     alert(TheJavaScriptArray[i])
   }
}
```

It should be noted that you are not able to create a VBArray in JavaScript. This is one of the reasons that the method GetExpectedUtility2 was added to the control. The original method (GetExpectedUtility) expected you to pass it a VB Array as its second parameter. GetExpecetedUtility2 accepts a comma-separated list (string) instead.

B.4.2 Accessing the control through scripting

Adding the control

Adding the control in your HTML-code can be done in exactly the same way as with VBScript:

```
<OBJECT
   CLASSID="CLSID:035C8025-AC03-11D2-B144-0000C0E59BCA"
   CODEBASE="http://www.whatever.edu/ActiveX/SmileX.ocx"
   ID=SmileX
   HEIGHT=0
   WIDTH=0>
</OBJECT>
```

See Section 3.1 for more details.

Accessing properties

A property can be accessed in JavaScript as follows:

```
NetID = SmileX.NetworkID ;
```

This will assign the network ID to the variable NetID, assuming that a network-file has been loaded into the control. The only difference with VBScript is the semicolon (';') behind the statement.

Using Methods of the control

Methods can be invoked as follows:

```
Result = SmileX.ReadFile("c:\\smilex\\animals.dsl") ;
```

This loads the file 'c:\smilex\animals.dsl' into the control. And assigns the resulting value to the variable Result. Again a semicolon is added. If a method requires more than one parameter, separate the parameters by a comma:

```
Result = SmileX.SetEvidence(Node0,Choice1) ;
```

B.4.3 Using HTML INPUT values

This section describes some usefull HTML controls that you can use in pages where you use the SmileX control.

Textbox

```
<INPUT TYPE=TEXT NAME=MyTextBox VALUE="">
```

This adds a textbox to the HTML-page which can be referred to as MyTextBox. It's initial value will be the string " (without the quotes). Use the Value property of the textbox to pass the current value to a function or to change it.

```
document.theform.MyTextBox.value = _
  SmileX.GetEvidence("MyNode0") ;
```

This bit of code finds the status or choice that is set as evidence for the node MyNode0 and displays it in the textbox MyTextBox. Note that (as opposed to VBScript) the variable MyTextBox should be prefixed by `document.theform` , that is, if the object MyTextBox is in a form with the name `theform`. Also note that the `value` has to be with a lowercase 'v'. To use the value entered in a textbox as a parameter to a method of the SmileX control use the following syntax:

```
SmileX.ReadFile(document.theform.MyTextBox.value) ;
```

This statement reads a file into the control. The parameter filename is retrieved from the textbox. Note that in this example the result (return value) is not recorded in a variable. In general you should check the return value for possible errors.

Button

A button is often used in HTML-pages to perform a certain action. Sometimes it will submit values and bring you to another page, but this is not obligatory. You can also use a button to invoke functions of the SmileX control. A button is inserted into a HTML page as follows:

```
<INPUT TYPE=BUTTON VALUE="OK" NAME="MyButton"
       ONCHANGE="MyButtonClick()">
```

This will insert a button into your page with the caption 'OK' (no quotes) and the name MyButton. To make the button execute a JavaScript function when it is clicked, do not forget the `ONCLICK` value. Note that the parentheses have to be added to the function-name, if you do not do it, the function will not be called. The JavaScript function that is invoked by pushing this button could look like this:

```
function MyButtonClick()
{
  SmileX.UpdateBeliefs("Lauritzen")
}
```

As opposed to Visual Basic Script there is no naming convention for these functions, you simply add the ONCLICK tag to the button definition and specify there which function you want to call.

Radio buttons

When the user can make one choice out of a small number of options it may be usefull to use radio buttons. These buttons can work as a group and enabling one of them will automatically disable the rest of the buttons in the group. An example:

```
 <FORM NAME=theform>
   <INPUT TYPE=RADIO NAME=SEX VALUE="Male"
          ONCLICK="SexClick()" > Male<BR>
   <INPUT TYPE=RADIO NAME=SEX VALUE="Female"
          ONCLICK="SexClick()"> Female<BR>
</FORM>
```

This inserts a group of two radion buttons (the buttons work as a group because they have the same NAME property) of which the user can choose one. Using the value of the group can be done as follows:

```
function SexClick()
{
  for(i=0;i<document.theform.Sex.length;i++)
  {
    if (document.theform.Sex[i].checked)
    {
      SetEvidence("MyNode",document.theform.Sex.value) ;
    }
  }
}
```

This sets the evidence specified by the choice in the radio button group for the node MyNode. Whenever the user clicks on a radio button in the group, the function SexClick() will be fired. For some strange reason the values of the radio buttons are not stored in the order they are declared. In the example above

`document.theform.Sex[0].value` is 'Female' `document.theform.Sex[1].value` is 'Male'

Although the radiobutton for 'Male' was declared first in the HTML code, it shows up in the last place of the array of values. It turns out that this is always the case and that the values are always backwards.

Checkbox

A checkbox can be used when you want the user to be able to choose between a True or False value. You can for example let the user decide whether or not to use a certain node as a target-node:

```
<INPUT TYPE=CHECKBOX NAME=MyCheckBox
 VALUE="Use Mynode as target" ONCLICK="MyCheckBoxClick()">
```

This creates a checkbox with the caption 'Use Mynode as target'. This chaeckbox can be referred by its name MyCheckBox. To use the status of to determine whether or not to use the node as a target requires a statement similar to the following:

```
if (document.TheForm.MyCheckBox.selected)
{
  SmileX.SetTarget("MyNode")
} else {
  SmileX.ClearTarget("MyNode")
}
```

Executing this statement whenever the status of the checkbox is changed can be done by putting the code in the function specified in the ONCLICK tag:

```
function MyCheckBoxClick()
{
  if(document.TheForm.MyCheckBox.selected)
  {
    SmileX.SetTarget("MyNode")
  } else {
    SmileX.ClearTarget("MyNode")
  }
}
```

Combobox

A combobox is often used when the user can make a choice out of a limited number of options. One will often find a combobox in pages where the country of residence has to be supplied by the user, the combobox then contains a list of (specified) countries. Inserting a combobox into a HTML page can be done as follows: (notice that this is not really an INPUT object, but a SELECT object!)

```
<FROM NAME=TheForm>
<SELECT NAME=MyCombobox ONCHANGE="MyComboBoxChange()">
      <OPTION VALUE="Mammal" SELECTED>A Mammal
      <OPTION VALUE="Reptile">A Reptile
      <OPTION VALUE="Bird">A Bird
      </SELECT>
</FORM>
```

This inserts a combobox with tree choices into the page. The choices need to be specified by a value and a name. The initial value can be set by using the SELECTED tag in an OPTION VALUE declaration. Every option is followed by the string that is actually displayed in the combobox on the screen. In this case the option value Mammal displays the words 'A Mammal'. Using the current selected value of the combobox can be done as follows:

```
SmileX.SetEvidence("AnimalClass", _
  document.TheForm.MyCombobox.value)
```

This will set the class chosen in the combobox as evidence for the node AnimalClass. It may be usefull sometimes to perform an action whenever the user click (changes) the value in the combobox. To do this enter a function name in the ONCLICK tag, as in the example. The functino itself could look like this:

```
function MyComboBoxChange()
{
  SmileX.SetEvidence("AnimalClass", _
    document.TheForm.MyCombobox.value)
}
```

With JavaScript it is very easy to change the contents of a combobox. You even declare an emtpy list of options in HTML and fill the itemlist later in your Java Script block. An Example:

```
<FORM NAME=TheForm>
  <SELECT NAME=MySelect">
  </SELECT>
</FORM>
<SCRIPT LANGUAGE=JavaScript>
  var IPVB = SmileX.GetAllNodes();
  if (IPVB!=null)
  {
    var IP=IPVB.toArray();
    document.TheForm.MySelect.length = IP.length;
    for (i=i;i<IP.length;i++)
    {
     document.TheForm.MySelect.options[i].value = IP[i];
     document.TheForm.MySelect.options[i].text = IP[i];
    }
  }
</SCRIPT>
```

This piece of code fills the combobox MySelect with the nodenames of the loaded network as items.

B.5 Using SmileX with Excel

The first section of this chapter describes how to insert a control into your project, the rest of the chapter discusses some controls that may be use full to you and the most common properties and methods you may need.

B.5.1 Inserting controls into the Spreadsheet

Inserting the SmileX ActiveX control

Inserting a control into an excel spreadsheet can be done as follows:

1. Make sure the Control Toolbox is showing
   To show or hide the Control Toolbox Choose Toolbars from the View Menu and choose Control Toolbox in the submenu. The option you just choose lshould become checked (otherwise you have just toggled it off and you should repeat the actions to toggle it on again).

2. Enter Design Mode
   Click the Design Mode button in the top left corner of the Control Toolbox until it is down (depressed). This means that you have activated the Design mode in which you can add, remove or edit controls, Visual Basic Macro's etc.

3. Choose the SmileX control from the More Controls list.
   On the bottom right hand corner of the Control Toolbox you see a button that looks like a toolset. Click it and choose the entry SmileX Control from the resulting list.

4. Click on the appropriate place in the spreadsheet.
   Since the SmileX control does not have a display property it does not matter much where you place the control, since it won't be shown. Just put it in a place where you can find it back.

Inserting other controls

To insert other controls can be inserted in much the same way. You just choose another control from the list. Of course you can also choose other buttons in the toolbox, representing the most commonly used controls like Label, Textbox, ComboBox etc... These controls are actually the Microsoft Forms 2.0 controls, that, by the way, ship with Microsoft Office '97.

B.5.2 Using properties and Methods

Determinig the name of controls

Before you can start using the properties and methods of a control you first have to know how to make a reference to the specific control. The reference name (or name for short) of a control can be found in a lot of ways. We describe two:

1. Doubleclicking the control
   This will start the Visual Basic for Applications environment. The cursor will be placed in a subroutine (since you doubleclicked the control this will

usually be the GotFocus event handler of the object). The name of the subroutine contains the name of the control you doubleclicked (the part before the underscore).
In most cases the name of the control in the spreadsheet is just the name of the sort of control you inserted (the control-class) with a number added at the end, starting from 1 (one)
Example:

```
Private Sub SmileX1_GotFocus()

End Sub
```

The name of the control is in this case SmileX1.

2. Using the properties window
Select the control in the spreadsheet and choose the properties button from the Control Toolbox (usually the second button from the left in the top row). This will display the properties window of the currently selected control. The combobox on the top of that window will contain the control name and the control class name (in that order).

```
SmileX1 SmileX
```

The name of the control in this example is again SmileX1.

Accessing properties and methods of the controls

Accessing the properties and methods of the controls can be done in pretty much the same way as described in chapter 4. There are some minor differences between Visual Basic for Applications (as used in Excel) and Visual Basic Script but this will not be really important in using the SmileX and other controls.

An Example:
Start a fresh Excel spreadsheet and insert a SmileX control and a button control Type the word 'Bird' in cell A1 (without quotes). Doubleclick the button, which will bring you to the OnCLick event-handler of the button. Complete the code of the function as done here:

```
Private Sub CommandButton1_Click()
  Workbooks("Book1").Sheets("Sheet1").Range("A3") = _
     SmileX1.ReadFile("c:\smilex\animals.dsl")
  Workbooks("Book1").Sheets("Sheet1").Range("A4") = _
     SmileX1.NetworkID
  SmileX1.SetEvidence "Class", _
     Workbooks("Book1").Sheets("Sheet1").Range("A1")
```

```
  Workbooks("Book1").Sheets("Sheet1").Range("A2") = _
     SmileX1.GetEvidence("Class")
End Sub
```

This example assumes that you did not change the names of the workbook or sheets in the spreadsheet (i.e. the name of the file is 'Book1' and the name of the first sheet is 'Sheet1') and that the names of the controls are SmileX1 and CommandButton1. It also assumes that the file animals.dsl is in the directory as mentioned in the first line.
When you return to Excel (ALT+Q) and exit Design Mode (Click the Design Mode button in the Control Toolbox to turn it off) you can click the button and see what happens.
In this case the result of the call to the ReadFile Method will be placed in the cell A3 and the ID of the network loaded into the control will be placed in the cell A4.
Then the word Bird will be read from cell A1 and set as evidence for the node Class in the animals network. Finally the evidence just set will be retrieved from the control and put in cell A2.

B.5.3 Handling arrays

Handling arrays can be rather difficult, but with the examples provided in this section you should be able to handle them correctly.

Displaying the values of an Array

When a SmileX (or other) method returns an array as a result you want to be able to display the values in this array. Instead of using the named cells we now choose another way of referring to cells. Check out this Example:

```
Private Sub CommandButton1_Click()
  Dim Index
  SmileX1.ReadFile ("c:\smilex\animals.dsl")
  MyOutcomes = SmileX1.GetOutcomes("Class")
  Index = 1
  If Not IsEmpty(MyOutcomes) Then
    For Each Outcome In MyOutcomes
      Workbooks("Book1").Sheets("Sheet1").Cells(Index, _
                1).Value = Outcome
      Index = Index + 1
    Next
  End If
End Sub
```

This example reads the file animals.dsl finds the outcomes and puts the outcomes in cells A1 to A3 (Node 'Class' has 3 outcomes. As you see cells can also be referenced with the Cells function. The first parameter determines the number of the cell and the second determines the letter of the cell-name (A=1, B=2 and so on). These numbers can also be a variable like the first parameter in the example, which enables you to store the separate values in the array in separate cells.

Putting cell-values into an array

In this section we will show you how to create an array. The best thing to do is to look at an example right away.

```
Private Sub CommandButton1\_Click()
  NumOfIndexingParents = 3
  SmileX1.ReadFile ("c:\smilex\pinball.dsl")
  SmileX1.UpdateBeliefs ("Lauritzen")
  Dim MyOutcomes As Variant
  MyOutcomes = Array("", "")
  ReDim Preserve MyOutcomes(NumOfIndexingParents)
  For TheIndex = 1 To NumOfIndexingParents
    MyOutcomes(TheIndex - 1) = _
    Workbooks("Book1").Sheets("Sheet1").Cells(TheIndex, 1).Value
  Next
  Result = SmileX1.GetExpectedUtility("TotalRevenue", MyOutcomes)
  If Not IsEmpty(Result) Then
    Workbooks("Book1.xls").Sheets("Sheet1").Range("B1") = Result
  End If
End Sub
```

This example creates an array with the correct size and gets the strings for the array from the cells A1-A3 (enter for example the following values in the cells(without the quotes): A1:'Vending' A2:'Bar' A3:'High'.) The value of cell B1 will be the expected utility of node 'TotalRevenue' for the outcomes specified in the array, that is, if the result is not empty, which means that an error occured in the method GetExpectedUtility. Do not forget to declare the variable you want to assign an array to as a Variant first.

B.6 Using other controls

This chapter describes the use of some other commonly used controls. Especially the Microsoft Forms 2.0 controls are widely used, you can for example use them in Excel spreadsheets, Word documents etc. We will discuss the basic features of these controls and show how they can work together with SmileX in some basic examples.

The other controls discussed in this chapter are excellent for use with SmileX. The examples will show you why and how they can be used.

B.6.1 Using Microsoft Forms 2.0 controls

In this section you will find information and examples of how you can use Microsoft Forms 2.0 Controls together with the SmileX control in your project. We do not try to be complete in this section, but we rather focus on some basic (but often-used) examples. The Forms 2.0 package comes with Microsoft Office 97. The package contains more controls than described here. More information on these controls can be found on the Microsoft website.

Label

A label can be used as a dynamic piece of text, that is, a piece of text that can change depending on certain events or actions. inserted as follows: The most important property of a label is of course its caption:

```
MyLabel.Caption = SmileX.NetworkID
```

Textbox

A Forms 2.0 Textbox is similar to a HTML textbox. It shows a box where text can be inserted and edited in. The Value property is the most important property of this control:

```
SmileX.SetTarget(MyTextBox.Value)
```

ComboBox

The main difference between Forms 2.0 controls and HTML controls is the dynamic behaviour of the Forms 2.0 controls. With HTML controls you can't change the entries in a combobox, with ActiveX however, you can! The following example shows you how you can load entries (in this case node-names into the control. Before loading begins all previous entries are removed from the list. This makes sure that the names will not be just added to the list, keeping the old entries as well.

```
For Index = 1 To MyComboBox.ListCount
  MyComboBox.RemoveItem(0)
```

```
Next
' Make the value an empty string
MyComboBox.Value=""
AllNodes = SmileX.GetAllNodes(0)
If Not IsEmpty(AllNodes) Then
  For Each Node In AllNodes
    MyComboBox.AddItem Node, MyComboBox.ListCount
  Next
  ' Select the first element
  MyComboBox.ListIndex = 0
End If
```

ListBox

A listbox is more or less the same as a Combobox. The only (obvious) difference is in displaying it. You can use it as follows:

```
For Index = 1 To MyComboBox.ListCount
  MyListBox.RemoveItem(0)
Next
AllNodes = SmileX.GetAllNodes(0)
If Not IsEmpty(AllNodes) Then
  For Each Node In AllNodes
    MyListBox.AddItem Node, MyListBox.ListCount
  Next
  ' Select the first element
  MyListBox.ListIndex = 0
End If
```

Using the (selected) value in the listbox can be done like this:

```
SmileX.SetTarget(MyListBox.Value)
```

Other controls in the Forms 2.0 Package

There are a lot more controls that you can use in your project in the Microsoft Forms 2.0 Package. Here is a list of All the available controls with their CLASSID. You will need the CLASSID to insert these controls into a web-page. Other environments offer a drop-down-list of the available controls usually.

B.6.2 Other usefull controls

Progress Bar

The progress-bar can be used to visualise probablities or other (numerical) values. The CLASSID of the control is 35053A22-8589-11D1-B16A-00C0F0283628A which can be used to include the control into a HTML-page. The most important properties of the control are the Min, Max en Value property.

Control	**CLASSID**
CheckBox	8BD21D40-EC42-11CE-9E0D-00AA006002F3
Combobox	8BD21D30-EC42-11CE-9E0D-00AA006002F3
CommandButton	D7053240-CE69-11CD-A777-00DD01143C57
Frame	6E182020-F460-11CE-9BCD-00AA00608E01
Image	4C599241-6926-101B-9992-00000B65C6F9
Label	978C9E23-D4B0-11CE-BF2D-00AA003F40D0
Listbox	8BD21D20-EC42-11CE-9E0D-00AA006002F3
MultiPage	46E31370-3F7A-11CE-BED6-00AA00611080
OptionButton	8BD21D50-EC42-11CE-9E0D-00AA006002F3
Scrollbar	DFD181E0-5E2F-11CE-A449-00AA004A803D
SpinButton	79176FB0-B7F2-11CE-97EF-00AA006D2776
TabStrip	EAE50EB0-4A62-11CE-BED6-00AA00611080
TextBox	8BD21D10-EC42-11CE-9E0D-00AA006002F3
ToggleButton	8BD21D60-EC42-11CE-9E0D-00AA006002F3

Table B.2: Forms 2.0 controls and their CLASSID's

```
Sub UpdateProgressbars
  Result = 100 * SmileX.GetBeliefForOutcome("Animal","Monkey")
  If Result>0 And Result
```

This example finds the probability of the animal is a Monkey. This probability is then dsiplayed as the value of the progress-bar. A full progress-bar means in this case that the animal is a monkey with a 100

Slidebar

A slidebar can be used to have the user input a value in a certain range. This range can be set by the properties Min and Max.

```
  MySlideBar.Max = 10000
  MySlidebar.Min = 0
  MySlideBar.Value = 500
```

With the value-property you can (pre-)set the value of the slidebar. The last line in the example above places the value-thumb in the middle. The user can adjust it. When you need to perform an action when the user changes the value, you can use the Click even as follows:

```
Sub MySlideBar_Click
  If MySlideBar.Value < 200 Then
    Result = "Low"
  End If
  If MySlideBar.Value < 700 And MySlideBar.Value > =20 Then
    Result = "Medium"
  End If
  If MySlideBar.Value > = 700 Then
```

```
    Result = "High"
  End If
  res = SmileX.SetEvidence("Setting", Result )
  If res = 0 Then
    CurrentSetting.Value = Result
  Else
    CurrentSetting.Value = "Could not set the evidence"
  End If
End Sub
```

This sets the appropriate evidence for the node Setting in the Pinball network. Depending on the value of the slidebar, the subroutine picks a setting and sets this as evidence. Of course you can use a combobox or a plain text-field to let the user input a setting but sometimes a slidebar is just more intuitive. The CLASSID of this control is:373FF7F0-EB8B-11CD-8820-08002B2F4F5A.

B.7 SmileX FAQ and Troubleshooting

This appendix discusses some common problems and frequently asked questions.

B.7.1 Troubleshooting

I can't seem to catch the OnChange event for a HTML radio button group

That is a known problem. We are also having trouble with it. The HTML specifications seem to contradict eachother at this point. We have not yet found a solution for this problem.

I can't find the control

If you can't find SmileX in your registry then you may have forgotten to register the control. You will not find the control in other lists of available controls either then, so make sure you register the control. By the way: registering the control does not mean that you should contact us and pay money. It only means that you should make your computer 'aware' that it holds the control. How you can register the control is discussed in the chapter about web-pages.

B.7.2 Frequently Asked Questions

Why is there no JavaBean version of Smile?

The answer to this question is simple: because we didn't make it. We realise that JavaBeans are (also) very widely used and have certain advantages over ActiveX controls. We also realise however that ActiveX controls have certain advantages over JavaBeans.

Will SmileX eventually support all the functionality in Smile?

Probably not. The main reason for the the development of SmileX was to offer the possibility to create custom applications using Smile technology to support Decision Support. Currently SmileX is a mere wrapper around the Smile C++ classes and implements only the functions needed for extracting Decision support information from an existing network (created with Genie for example).

Hey, 'SmileX' isn't that...

Yep, it is the name of the gas used by the Joker in a Batman movie.

B.7.3 More Information

Smile and Genie

More infotmation about Smile and GeNIe can be found at the webpages of the

- Decisions Systems Laboratory
- Decision Systems Laboratory

- Genie and Smile Homepage

Visual Basic Script

More information on Visual basic can be found at:

- Microsoft VB Script language reference

HTML

HTML specifications can be found at:

- World Wide Web Consortium

JavaScript

- Microsoft JavaScript language reference
- Coolnerds JavaScrip reference and examples

[]

Bibliography

[1] Jerry Anderson. *ActiveX Programming with Visual C++ 5.0*. QUE Corporation, 1997.

[2] Unkown Author. Loan risk ratings, classification systems and loss foactors. *Journal of Commercial Lending*, 76(9):57–64, May 1994.

[3] R.H. Behrens. *Commercial Problem Loans*. Bankers Publishing Company, second edition edition, 1983.

[4] R.S. Canter. Lender beware: Federal regulation of consumer credit. *Credit World*, 81(5):16–21, May/June 1993.

[5] J.B. Caouette, E.I. Altman, and P. Narayanan. *Managing Credit Risk, The Next Great Financial Challange*. John Wiley & Sons, Inc., 1998.

[6] David Chappell. *Understanding ActiveX and OLE, a Guide for Developers and Managers*. Strategic Technology Series. Microsoft Press, 1996.

[7] Robert T. Clemen. *Making Hard Decisions, an Introduction to Decision Analysis*. Duxbury Press, 1991.

[8] G.F. Cooper. *NESTOR: A Computer-based Medical Diagnostic Aid that Integrates Causal and Probabilistic Knowledge*. Stanford University, 1984.

[9] Adam Denning. *ActiveX Controls Inside Out*. Microsoft Programming Series. Microsoft Press, second edition, 1997.

[10] L.W. Glorfeld and B.C. Hardgrave. An improved method for developing neural networks: The case of evaluating commercial loan creditworthiness. *Computers & Operations Research*, 23(10):933–944, 1996.

[11] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, (15):225–263, 1996.

[12] Kamran Husain. *ActiveX Developer's Resource*. Prentice Hall PTR, 1997.

[13] D. Korn. What's your score? *Black Enterprise*, 29:89, August 1998.

[14] W.B. McNeill. *Primary Areas of Consumer Credit*, chapter 57, pages 750–767. Dow Jones-Irvin, 1978.

[15] R.A. Nunlist, J.C. Seibert, and C.R. Turner. *Industrial and Consumer Credit Management*. Grid Publishing, Inc., 1983.

[16] K.L. Parkinson. Why credit managers should be concerned about consumer credit laws: A conversation with charles tatelbaum, esq. *Business Credit*, 100(4):18–20, April 1998.

[17] J.B. Quinn. They've got your number. *Newsweek*, 128:44, March 1996.

[18] M.L. Rechkemmer. *Establishing a Loan Review Department*, chapter 2, pages 15–28. Bankers Publishing Company, second edition edition, 1983.

[19] Dale Rogerson. *Inside COM*. Microsoft Programming Series. Microsoft Press, 1997.

[20] S.G. Romaniuk and L.O. Hall. Decision making on creditwortiness using a fuzzy connectionist model. *Fuzzy Sets and Systems*, 48:15–22, 1992.

[21] J. Seiberg. Bank hits plan to let them collect small-business race and sex data. *American Banker*, 160(162):2, August 1995.

[22] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, second edition, 1991.

[23] Author Unknown. Fair lending. *Bank Marketing*, 27(1):60–62, Januari 1995.

[24] J. Zack. System lets first fidelity monitor loan risk. *American Banker*, 160(63):14, April 1995.

[25] M. Zmiewski and B. Foster. Creditscoring speeds small business loan processing. *Journal of Lending and Credit Risk Management*, 79(3):42, November 1996.

C.P.R.Thijssen, October 10, 1999