Technical White Paper

The HUGIN FRAUD DETECTION MANAGEMENT (FDM) SOLUTION

HUGINEXPERT
The leading decision support tool

HUGIN Fraud Detection Management – Technical White Paper

Table of Contents:

1 Introduction

This white paper introduces the HUGIN Fraud Detection Management (FDM) solution, and gives an overview of the tasks involved in developing and implementing a fraud detection system based on HUGIN FDM from a technical and functional point of view. In addition to the technical and functional properties of the HUGIN FDM, this white paper also provides examples of solution integration using the HUGIN Java API.

The target readers of this document are actuaries and IT staff members involved in the model development and system integration process.

HUGIN FDM is an efficient solution for detecting fraudulent insurance claims based on advanced statistical graphical models. This document describes the process of developing and integrating a HUGIN fraud detection model (nonlife) into an existing IT platform using the HUGIN Java Application Programming Interface (API).

Figure 1 below illustrates the use of the HUGIN FDM system in a nonlife claims handling process. The claims handler receives a claim and collects information on the claim. This information is entered into the claims handling system and combined with any information the company has on the client. To support the decision on whether to direct the claim to the fast track for payment and closing or to hand the claim over to a claims investigator, the user interface has a simple “traffic light”. The “traffic light” is a decision support system controlled by the HUGIN FDM. The traffic light improves the claims handler’s ability to make faster, more informed and consistent decisions on the likelihood that a claim is fraudulent.

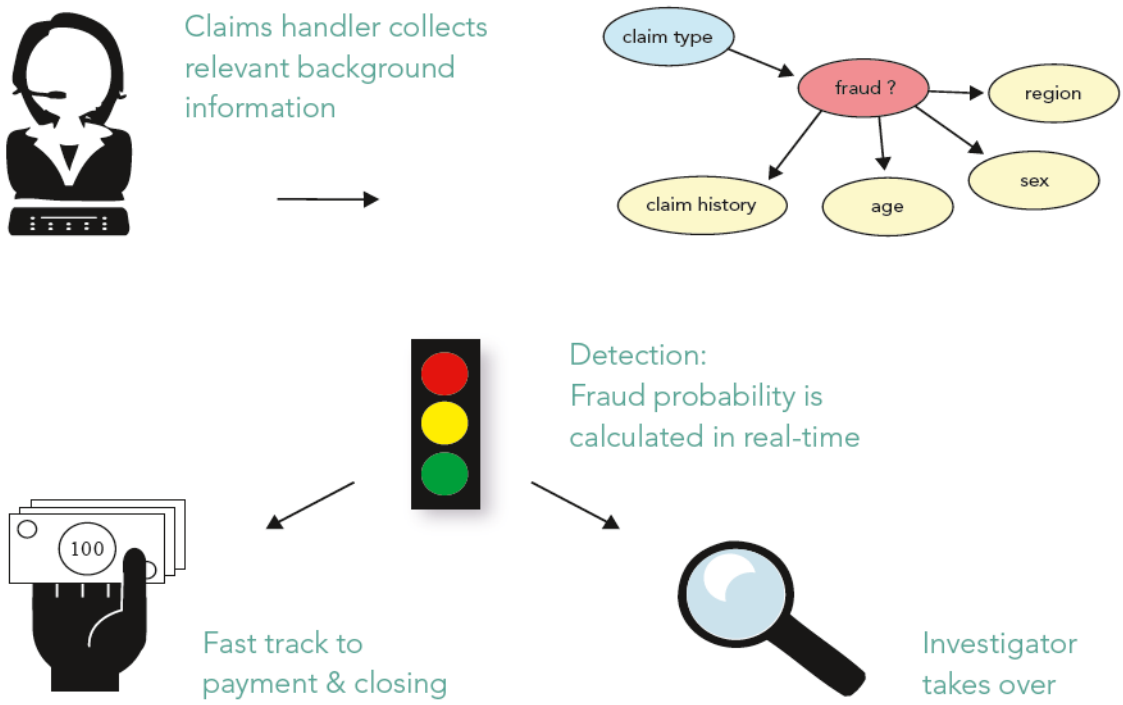


Figure 1: The HUGIN FDM implemented as a traffic light on the screen of claims handlers

2 Example of a HUGIN FDM model

A central component in a model-based decision support system is the model. A fraud detection model specifies dependence relations between a fraudulent claim and a set of fraud indicators and risk characteristics, including customer behavior and customer attributes. In particular, a fraud detection model specifies which behavior and customer attributes differentiate a fraudulent claim from a legitimate claim.

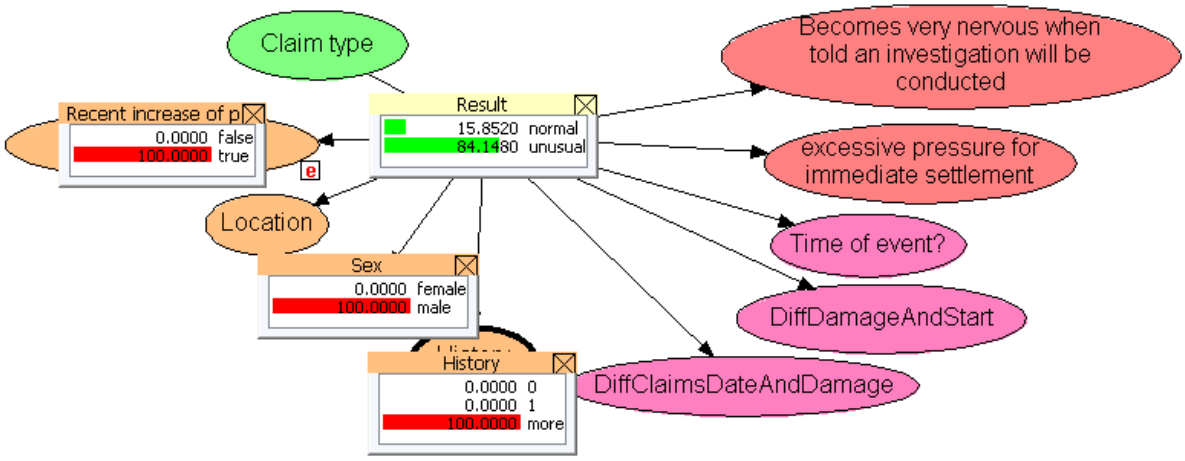
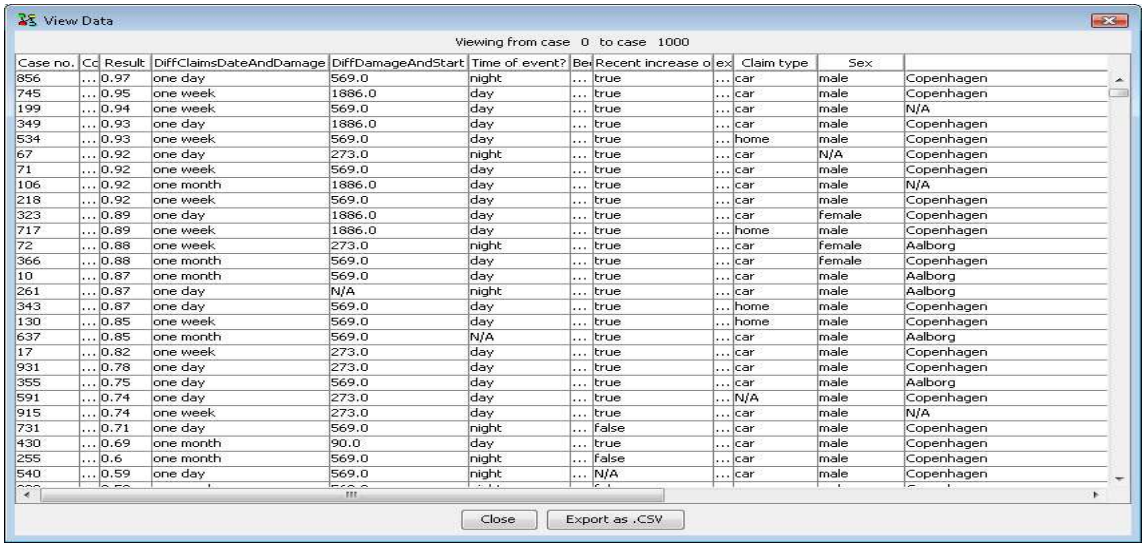


Figure 2: A simple HUGIN fraud detection model.

Figure 2 illustrates a simple fraud detection model. The model describes the dependence relations between claim type, fraud, claim history, age, etc. In the case of a burglary claim made by a male with a history of more than one claim who has recently increased the limits of his policy, the probability of the claim being fraudulent according to this model is 84.15%. Even with limited information the model can assess the probability of fraud. In this case, the given information indicates a high probability of fraud.

The HUGIN FDM solution can also be used to generate a list of claims sorted by fraud probability as shown in Figure 3. The list is in descending order according to the value in the "Result" column.


View Data

Viewing from case 0 to case 1000

Case no.	Cc	Result	DiffClaimsDateAndDamage	DiffDamageAndStart	Time of event?	Be	Recent increase o	ex	Claim type	Sex	
856	...	0.97	one day	569.0	night	...	true	...	car	male	Copenhagen
745	...	0.95	one week	1886.0	day	...	true	...	car	male	Copenhagen
199	...	0.94	one week	569.0	day	...	true	...	car	male	N/A
349	...	0.93	one day	1886.0	day	...	true	...	car	male	Copenhagen
534	...	0.93	one week	569.0	day	...	true	...	home	male	Copenhagen
67	...	0.92	one day	273.0	night	...	true	...	car	N/A	Copenhagen
71	...	0.92	one week	569.0	day	...	true	...	car	male	Copenhagen
106	...	0.92	one month	1886.0	day	...	true	...	car	male	N/A
218	...	0.92	one week	569.0	day	...	true	...	car	male	Copenhagen
323	...	0.89	one day	1886.0	day	...	true	...	car	female	Copenhagen
717	...	0.89	one week	1886.0	day	...	true	...	home	male	Copenhagen
72	...	0.88	one week	273.0	night	...	true	...	car	female	Aalborg
366	...	0.88	one month	569.0	day	...	true	...	car	female	Copenhagen
10	...	0.87	one month	569.0	day	...	true	...	car	male	Aalborg
261	...	0.87	one day	N/A	night	...	true	...	car	male	Aalborg
343	...	0.87	one day	569.0	day	...	true	...	home	male	Copenhagen
130	...	0.85	one week	569.0	day	...	true	...	home	male	Copenhagen
637	...	0.85	one month	569.0	N/A	...	true	...	car	male	Aalborg
17	...	0.82	one week	273.0	day	...	true	...	car	male	Copenhagen
931	...	0.78	one day	273.0	day	...	true	...	car	male	Copenhagen
355	...	0.75	one day	569.0	day	...	true	...	car	male	Aalborg
591	...	0.74	one day	273.0	day	...	true	...	N/A	male	Copenhagen
915	...	0.74	one week	273.0	day	...	true	...	car	male	N/A
731	...	0.71	one day	569.0	night	...	false	...	car	male	Copenhagen
430	...	0.69	one month	90.0	day	...	true	...	car	male	Copenhagen
255	...	0.6	one month	569.0	night	...	false	...	car	male	Copenhagen
540	...	0.59	one day	569.0	night	...	N/A	...	car	male	Copenhagen

Close Export as .CSV

Figure 3: A list of claims sorted according to fraud probability

Figure 3 shows one type of list that can be generated using HUGIN Graphical User Interface (GUI). Similar functionality can easily be integrated into existing IT platforms using a HUGIN API.

3 HUGIN Fraud Model Construction

The central component in a HUGIN FDM solution is the model. The model calculates the probability of fraud based on given information about the claim and the customer. In principle, the model can include any number of variables and have any graphical structure. Experience has shown that even a model with relatively few indicators and a simple structure as shown in Figure 2 can perform with impressive accuracy.

A HUGIN FDM model is constructed from expert experience and domain knowledge, or historical data, or a combination of the two sources. The typical process of developing a fraud model consists of the following sequence of steps:

1. Data collection and preparation

2. Network structure specification

3. Specification of expert knowledge and parameter estimation

4. Model evaluation

The HUGIN FDM system computes the fraud probability of a claim based on the knowledge represented in the fraud model and the information about the claim propagated in the model.

Each node in the fraud model represents a variable, usually specifying either the *target* variable (hypothesis or classification variable) which is the variable indicating whether or not a claim is fraudulent or suspicious, or an *indicator* variable, which is an attribute of the claim or the customer used to differentiate between fraudulent and non-fraudulent insurance claims. A fraud model may also contain nodes that represent neither target nor indicator nodes.

Besides the simple structure illustrated in Figure 2, a fraud model can have any acyclic, directed graph structure. The structure in Figure 2, however, is the most common fraud model structure and experience has shown that this type of model is well suited for fraud prediction.

Figure 4 shows the model from Figure 2 in the HUGIN GUI with tables opened for the variables **“Sex”**, **“Recent increase of policy limits”** and **“DiffClaimsDateAndDamage”**. Each table represents the variables’ conditional probability given the variable “Result”. The entries of each table can be assessed either from expert domain knowledge and experience, estimated from historical data, or a combination of the two. For instance in Figure 4, the number 0.3 shown in the upper left corner of the **“Sex”** table is the conditional probability of “Sex=female” given “Result=normal”. This number can be assessed from expert knowledge and experience or estimated from data.

The model is a quantitative representation of the knowledge, experience and historical data an insurer has on insurance fraud. This knowledge, experience and data increases constantly as the insurance company operates and the system is used. Over time, an insurer will acquire experience using the system, and it will be necessary to integrate new data and knowledge into the model. We suggest revisiting the model at least once a year to make sure the system is up to date. This will ensure that any changes can be incorporated into the knowledge representation.

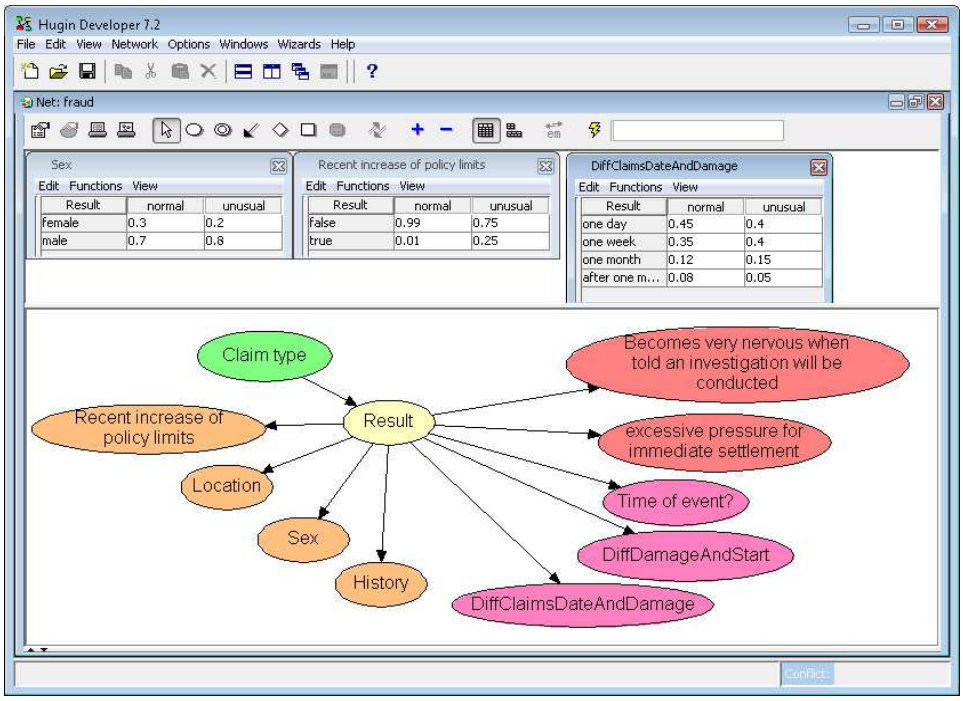


Figure 4: HUGIN FDM model

4 FDM Integration Example

This section describes how to integrate the HUGIN FDM into an existing IT platform. The two main tasks:

1. To link each variable in the fraud model to the appropriate data source
2. To read and display the probability of fraud once updated beliefs are computed

Integration of the HUGIN FDM solution into an existing IT platform is a simple task using a HUGIN API. Prior to integration, it is important to decide the probability of fraud will be displayed to the user. We suggest displaying fraud probability using a “traffic light” with colors red, yellow and green or only red and green. Integration consists of three steps:

1. Each indicator variable is linked to a data source in the existing IT platform
2. The belief of the state reflecting fraud in the target variable is linked to the “traffic light” (if traffic light display is used)
3. Implement a method to collect and enter relevant data, update beliefs using the fraud model, and display the result

Appendix A contains a complete example illustrating how to perform integration using the model shown in Figure 2. The example uses the HUGIN Java API and it consists of the following main steps:

1. Load the fraud model from file
2. Get handles to variables represented in the model
3. Compile the model into a computational structure
4. Enter observations from a sample claim
5. Propagate evidence to compute the probability that the claim is fraudulent
6. Display the results
7. Remove observations related to claim

Steps 1-3 need only be performed once. Steps 4 to 7 are repeated each time the fraud probability of a claim is calculated.

One of the main advantages of the HUGIN FDM solution is the clear separation between fraud model development, and the integration of the fraud model into an existing IT environment. Figure 5 illustrates the clear separation between model development, usually carried out by analysts and experts in the client organization, and system integration, usually carried out by in-house IT staff, sometimes with assistance from HUGIN consultants.

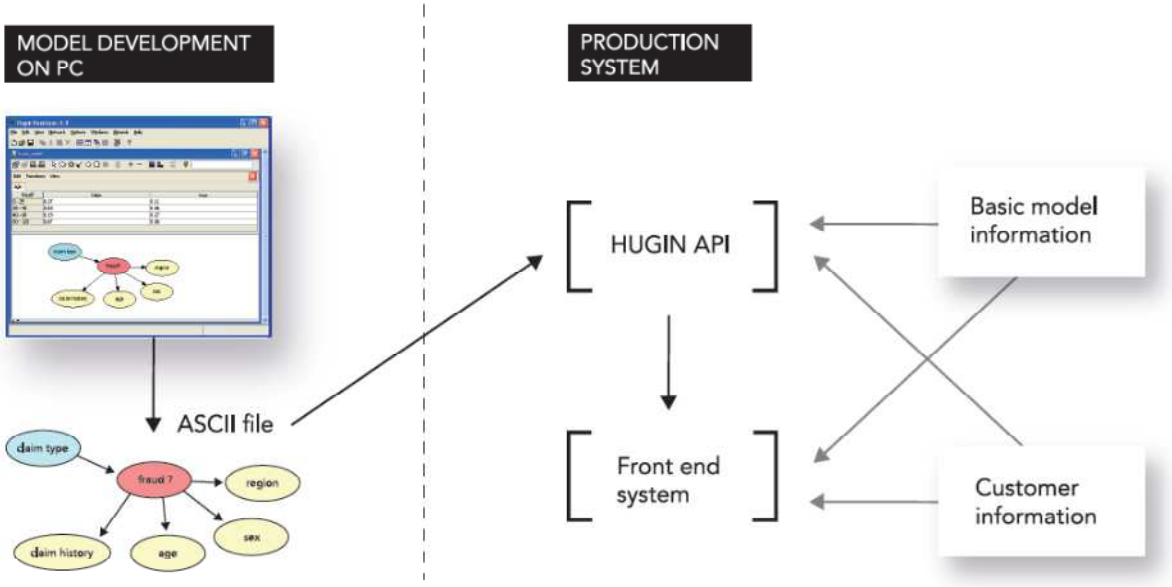


Figure 5: HUGIN FDM offers clear separation between model development and system integration

Once integration has been completed, it is a simple task to replace an existing model with a new one. If no new variables are added to the model, updating a model consists of replacing a single ASCII text file. If a new variable is added to the model, then this variable has to be linked to the appropriate data source. That is it.

See chapter 1 of the HUGIN API Reference Manual for further technical details on the use of HUGIN APIs with different software and hardware platforms, including information on how to compile and execute applications using HUGIN API functionality.

5 The Technology

The technology behind the HUGIN FDM solution is called Bayesian Networks (also known as Bayesian Belief Networks, causal probabilistic models, Bayes nets, etc.). Bayesian networks are complex diagrams that organize the body of knowledge in a given area by mapping out cause and effect relations among key variables and assigning them with numbers that represent the extent to which one variable is likely to affect the other. This technology and HUGIN software have a long list of advantages over competing systems:

- It combines expert knowledge and historical data.
- It handles missing values in data and computes with partial observations.
- Efficient inference engine making real time inference possible.

- Models and implementations are flexible and easy to maintain, extend and revise.
- Models are easy to integrate into existing systems.
- Intuitive graphical models make it easy to communicate about fraud risk and model properties.

The technology and HUGIN software also support various types of analysis, for example, value of information analysis, parameter sensitivity analysis, evidence sensitivity analysis and data conflict analysis. These analysis methods can be helpful both in fraud model development and system usage.

6 Functional information

For easy reference and to provide the most useful information, the next 2 sections 6 and 7 are organized as question and answer sessions.

6.1 Are Fraud Models available for all insurance segments?

A HUGIN fraud detection model is developed in cooperation between domain experts such as claims handlers and SIU agents and a HUGIN consultant. Models can be developed for all insurance segments where there is access to domain expertise and/or historical data.

A common approach is to develop a model for one segment and use this model as a starting point for developing models for other segments. The technology and tool are very flexible and support an iterative development process in which models are continuously refined and extended.

6.2 Please describe the fraud detection options in the claims handling process

A HUGIN fraud model describes probabilistic dependence relations between a "fraud" variable and a set of indicator variables such as sex, age, and claims history. The model computes the probability of fraud for a given claim in real time. The model computes with partial information (missing values). Thus, it is possible to compute the probability of fraud in real time each time new information is received, and the tool can suggest appropriate questions if represented in the model.

The processing time depends on the complexity of the model, but for the most common types of fraud, model time and space complexity is linear in the number of indicator variables. The fraud detection model is a probabilistic graphical model (a Bayesian network) and it is usually static and updated only a few times a year.

The HUGIN GUI has functionality for both manual refinement, and re-estimation of the model from data and expert knowledge. Once the model is integrated into an existing IT platform, it is only necessary to make additional integration efforts when the model is extended with new variables. There is a clear separation between model development and model integration.

6.3 Does the tool offer additional workflow support for the claims handling process?

The HUGIN tool supports conflict analysis to determine "surprising" observations, and value of information analysis to determine the most "valuable" next observations, such as questions to the client. The processing time of computations are based on the complexity of the model.

6.4 Which data analysis capabilities / investigation support are available?

The HUGIN Graphical User Interface has different data-analysis capabilities for identifying correlations between variables, and estimating the strength of correlations between variables from historical data. . The tool supports sensitivity analysis, value of information analysis and conflict analysis, the last of which supports the identification of "surprising" sets of observations

7 Technical information

7.1 Please describe the software

The HUGIN software package consists of the HUGIN Graphical User Interface for fraud model development and the HUGIN Decision Engine for fraud model integration. The HUGIN Decision Engine has APIs for C, C#, C++, Java and VBA. The HUGIN GUI is a Java implementation (using JNI) and the HUGIN Decision Engine is implemented using ISO C.

7.2 Please describe the data requirements

The HUGIN Graphical User Interface reads CSV files. The HUGIN Decision Engine has API functions for real time entering of data and reading data from file. The HUGIN tool does not store any data.

7.3 How is the tool integrated into the existing workflow?

The fraud detection model is integrated into an existing system using one of the APIs for the HUGIN Decision Engine. APIs are available for C, C#, C++, Java and VBA. The HUGIN Decision Engine contains functionality for entering information, computing the probability of fraud, and accessing the probability of fraud. The HUGIN FDM is usually implemented as a traffic light on the user interface of the claims handler. The HUGIN Graphical User Interface is only appropriate for analysts.

7.4 What are the technical requirements of the necessary interfaces?

The HUGIN Graphical User Interface is implemented in Java on top of the Java API for the HUGIN Decision Engine. The core of the HUGIN Decision Engine is implemented using ISO C for portability and efficiency reasons. The HUGIN Decision Engine has been deployed on a wide range of hardware platforms and operating systems, including PCs running Windows and Linux, Macs running Mac OS, PCs and SUN servers running Solaris, HP servers running HPUX, and IBM mainframes.

7.5 Please describe the performance of the tool

The HUGIN Decision Engine does not support user administration. The user of the HUGIN Decision Engine is responsible for the use of mutual exclusion variables, for instance if the model is to be accessed by multiple users at the same time. An alternative could be to have a dynamic pool of processes running the same model. The time and space complexity of the most common type of fraud detection model is linear in the number of indicator variables. The response time is hardware dependent.

7.6 Please describe the administration requirements of the tool

The HUGIN Graphical User Interface is installed on the desktop computer of the user, i.e., the analyst developing a fraud detection model. The HUGIN Decision Engine is distributed as a single file or a pair of files to be installed on the host computer. There is no user administration. A HUGIN FDM model is stored in a single file (usually as an ASCII file). This file can be accessed using the HUGIN Graphical User Interface and the HUGIN Decision Engine. If minor adjustments are made to an existing model, an update consists of replacing the file on the server.

7.7 How are software updates handled?

The HUGIN Graphical User Interface is a separate one-user program. The HUGIN GUI has functionality to automatically install new builds released on the HUGIN website. A new version of the HUGIN Decision Engine can be installed by replacing the existing files on the host. New versions of the software package are released 1-2 times each year. The functionality of the HUGIN Decision Engine is tested using a suite of testing programs. Updates are not mandatory. New updates are usually only installed on a host, when new functionality is required. This is rarely the case for fraud detection models.

Appendix A. Java example source code

```
/*
* This simple Java program illustrates the integration of a simple HUGIN Fraud Detection Model.
* The objective is to compute the posterior probability of fraud for a specific insurance claim.
* Information on the claim is entered to the model and the posterior belief of fraud is computed and
* displayed on standard output.
*
* The application is compiled with this command:
* javac -cp hapi71.jar;. FraudIntegration.java
* where hapi71.jar is the HUGIN Java API (version 7.1)
*
* The application is executed with this command (the dll/so file associated with the HUGIN Java
* API should be placed in the current directory):
* java -cp hapi71.jar;. FraudIntegration fraud.net where
* fraud.net is the HUGIN network defining the fraud model.
*
* The output on standard output should be: P(Result=unusual | evidence) = 76,05%
*
* Author: Anders L Madsen @ HUGIN EXPERT A/S
*
* For any questions or comments, please contact the author at: alm@hugin.com
*/
import COM.hugin.HAPI.*;
import java.text.DecimalFormat;

class FraudIntegration {
  // the model
  Domain dom = null;

  // the nodes / variables in the model
  LabelledDCNode Result, Sex, ClaimType;
  // ...

  // NumberedDCNode ...; // none in model

  IntervalDCNode DiffDamageAndStart;
  // ...

  BooleanDCNode PolicyIncrease;
  // ...

  // ContinuousChanceNode ...; // none in model

  public FraudIntegration (String name)
  {
      try {
         // load model from file. Done once.
        dom = new Domain (name, new DefaultClassParseListener ());

        // Get handles for the nodes of the model. One list of
        // nodes for each node type. Done once.
```

```
        // ContinuousChanceNodes ... none in the example

        // LabelledDCNodes
        Result = (LabelledDCNode)dom.getNodeByName("Result");
        Sex = (LabelledDCNode)dom.getNodeByName("sex"); // Notice that node name is unique, not
        //node label
        ClaimType = (LabelledDCNode)dom.getNodeByName("type");
        // ...

        // NumberedDCNodes
        // ...

        // BooleanDCNodes
        PolicyIncrease = (BooleanDCNode)dom.getNodeByName("increase");
        // ...

        // IntervalDCNodes
        DiffDamageAndStart = (IntervalDCNode)dom.getNodeByName("DiffDamageAndStart");
        // ...

    } catch (Exception e) {
        e.printStackTrace ();
        System.err.println (e.getMessage ());
    }
}

// Main procedure. This is where the actual calculations are performed.
protected void doCalculations () {
    DecimalFormat d = new DecimalFormat ();
    d.setMaximumFractionDigits (2);

    try{
        // compile the model for inference / analysis. Done once.
        dom.compile ();

        // link nodes (indicators) of model to data sources, use
        // node name as unique identifier. In the example data is
        // hard coded into the source code. The link between the
        // data source and HUGIN is made once.

        // in principle we should now have a loop over all cases,
        // but we only consider a single case
        // 1) insert information from case
        insertEvidence ();

        // 2) propagate evidence
        dom.propagate(Domain.H_EQUILIBRIUM_SUM, Domain.H_EVIDENCE_MODE_NORMAL);

        // 3) show belief of Result for the case
        System.out.println ("P(Result="+Result.getStateLabel(1) +" | evidence) = " +
                            d.format(Result.getBelief (1) * 100)
                            +"%");
```

```
        // 4) remove observations to prepare for next case
        dom.initialize ();

        // do steps 1-4 for all cases or each time the probability
        // of fraud is to be updated (evidence can be entered
        // incrementally. A propagation is required prior to
        // accessing the posterior belief of Result
    } catch (Exception e) {
        e.printStackTrace ();
        System.err.println (e.getMessage ());
    }
}

// It is very important that the data values entered matches the
// state labels/values. Otherwise they are ignored. State labels
// are case sensitive. Some values may need to be computed if not
// presented in the data source (e.g., age)
//
//
protected void insertEvidence () {
        try{
        // case: auto claim of male with recent increase in policy and claim made 186 days after policy
        //starts

        enterObservation (ClaimType, "car");
        enterObservation (PolicyIncrease, "true");
        enterObservation (DiffDamageAndStart, 186);
        enterObservation (Sex, "male");

        //... more data
    } catch (Exception e) {
        e.printStackTrace ();
        System.err.println (e.getMessage ());
    }
}

// There is one enterObservation method for each node type
protected void enterObservation (ContinuousChanceNode n, double value) {
     try{
        n.enterValue (value);
    } catch (Exception e) {
        e.printStackTrace ();
        System.err.println (e.getMessage ());
    }
}
// Notice that select state is case sensitive. getStateIndex returns -1 for unmathed values.
protected void enterObservation (LabelledDCNode n, String label) {
        try{
          n.selectState (n.getStateIndex (label));
```

```
      } catch (Exception e) {
         e.printStackTrace ();
         System.err.println (e.getMessage ());
      }
}

protected void enterObservation (BooleanDCNode n, String label) {
      try{
         n.selectState (n.getStateIndex (label));
      } catch (Exception e) {
         e.printStackTrace ();
         System.err.println (e.getMessage ());
      }
}

protected void enterObservation (NumberedDCNode n, double value) {
      try{
         n.selectState (n.getStateIndex (value));
      } catch (Exception e) {
         e.printStackTrace ();
         System.err.println (e.getMessage ());
      }
}

protected void enterObservation (IntervalDCNode n, double value) {
       try{
          n.selectState (n.getStateIndex (value));
       } catch (Exception e) {
          e.printStackTrace ();
          System.err.println (e.getMessage ());
       }
}

static public void main (String args[])
{
      if (args.length != 1) {
         System.out.println ("usage: <net>");
         System.exit (-1);
      }
      FraudIntegration fi = new FraudIntegration (args[0]);
      fi.doCalculations ();
 }
}
```

Appendix B. System performance

We have performed a series of experiments to assess and demonstrate the performance of the HUGIN FDM solution on two different examples of fraud detection models. Model M is a model with 18 indicators used by one of our HUGIN FDM customers, and Model N is the same model with 9 indicators.

To assess the performance of the HUGIN FDM, we have analyzed the time usage of estimating the parameters of the models from data sets of different sizes and computing the probability of fraud for each case in the same data sets.

The results of the experiment are shown in the tables below where time is specified in seconds. The experiments have been performed on a desktop PC (CPU i7 920, 12 GB RAM, Windows Vista) using the HUGIN Java API version 7.2 and Java 6 update 17.

The performance of the parameter estimation is shown in the table below (threshold value on number of iterations is zero and convergence threshold is 0.0001).

	Data set size			
Model	10000	50000	100000	500000
M (18)	1.08	5.39	10.89	54.45
N (9)	0.23	0.77	1.53	7.66

Table 1 : Run time performance of parameter estimation (in seconds).

The performance of the computing probabilities (also known as belief update) is shown in the table below. This includes entering each case as evidence into the model, propagation of evidence, and computing marginal beliefs.

	Data set size			
Model	10000	50000	100000	500000
M (18)	0.74	3.80	7.13	34.91
N (9)	0.37	1.83	3.65	18.15

Table 2: Run time performance of belief update (in seconds).

Data set has been randomly generated from the true model with 5% missing values (missing completely at random). The experiments illustrate the high performance of HUGIN FDM on a standard desktop PC. The time usage grows approximately linear with the size of the data sets of each model.

The conclusion is that parameter estimation of this type of model is very efficient and that large sets of cases can be analyzed in batch. Both batch and real time belief update is very efficient in the type of models used in this experiment. For instance, in model M the performance of belief update is

approximately 14.000cases/second, whereas the performance in model N is approximately twice the performance of model M.

Appendix C. Further information

Figure 6 illustrates the architecture of the HUGIN APIs. The HUGIN Decision Engine is implemented in ISO C. This makes the HUGIN Decision Engine highly portable and efficient. On top of the ISO C implementation, a layer of APIs is implemented. APIs are implemented for major programming languages such as C, C++, C#, Java and as an ActiveX Server for Visual Basic for Applications.

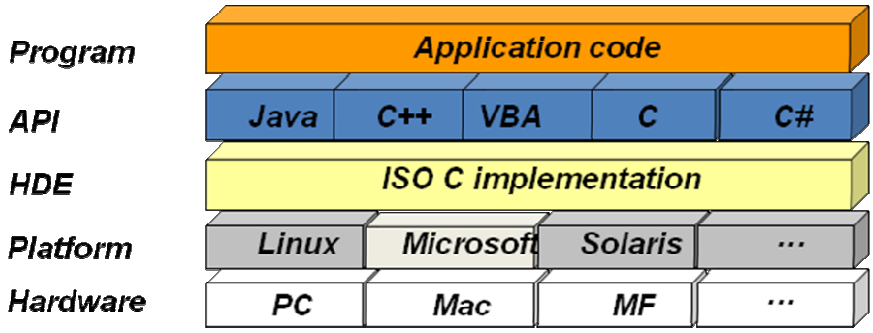


Figure 6: The HUGIN API architecture

Since the HUGIN Decision Engine is implemented in ISO C, it is highly portable. Instances of the HUGIN Decision Engine are known to run, or have run, on a number of different platforms including Linux, Microsoft Windows, SUN Solaris, et cetera and a number of different hardware platforms ranging from PDAs over Mac and PC to mainframe systems. The HUGIN Decision Engine runs on multi-user and multi CPU platforms and it available in both 32bit and 64bit versions.

Chapter 1 of the HUGIN API Reference Manual provides details on the use of the different HUGIN APIs on different platforms and hardware.