FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

EM4J - An Explanation Module Toolbox for Jess

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Sebastian Wurzer
Matrikelnummer 0426327

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.-Prof. Dr. Uwe Egly

Wien, 18.07.2011

(Unterschrift Verfasser)

(Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

EM4J - An Explanation Module Toolbox for Jess

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computational Intelligence

by

Sebastian Wurzer
Registration Number 0426327

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.-Prof. Dr. Uwe Egly

Vienna, 18.07.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Sebastian Wurzer
Joseph Haydn Gasse 16/6/20, 2514 Möllersdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Möllersdorf, 18.07.2011)	(Unterschrift Verfasser)

Acknowledgements

The research presented in this thesis has been supported by the Austrian Science Foundation (FWF) Project L514 *"i-MeaS - An Intelligent Image-Based Measurement System for Geo-Hazard Monitoring"*.
I want to thank my mother, Sonja, and father, Friedrich, that gave me the opportunity to study and their support through the years. Additionally I want to mention my friends that were always there to endorse me.
A special thank goes to my supervisor Uwe Egly for all his time, effort and patience. Last but not least I want to cherish my love Julia, the mother of our unborn child, for her endorsement and selfless endurance of the endless hours to fulfill this thesis.

Abstract

Explanations are an essential part of everybody's life. People continuously give and receive them in order to process information which in some cases would not be apprehend to them. Similarly, explanations are also a crucial part within the communication between knowledge based systems (KBSs) and their users. Every KBS needs to give explanations because of imprecise domains and the use of complex mechanisms. They are also needed to support the user in its decision making process whether the recommendation is valid or not. Decision makers have to trust the recommendation as well as the system itself. If they do not understand the process as well as the conclusion it is unlikely that they will accept it. Therefore explanations play a vital role in the acceptance of KBSs.

This thesis presents an explanation module toolbox (EM4J) for the Intelligent Geo-Risk Evaluation System (InGES), developed by the Institute of Geodesy and Geophysics and the Knowledge Based Systems Group (Institute of Information Systems) at the Vienna University of Technology, the Institute of Digital Image Processing at the Joanneum Research and the Institute of Geological Sciences at the University of Bern, Switzerland. Additionally EM4J supports the development of KBSs. The purpose of InGES is to compute risk levels for geological sites (e.g., hillsides). Since it works in a high failure cost environment the objective is to implement a highly technically transparent process oriented application highlighting the reasoning mechanism. This should support the user in accepting respectively refusing the recommendation given by InGES. EM4J is based on the approach of Wick et al. The design is process and verification oriented with a highly coupling of the inference engine. Therefore EM4J is tailored to a highly skilled user group. The development was divided into four phases - the implementation, the test, the integration and the evaluation.

Kurzfassung

Erklärungen sind ein integraler Bestandteil unseres täglichen Lebens. Menschen geben und erhalten sie kontinuierlich, um Informationen zu verarbeiten. In vielen Fällen ist diese Information für den Menschen sonst nicht wahrnehmbar. In ähnlichem Umfang sind Erklärungen ebenso ein essentieller Teil in der Kommunikation zwischen wissensbasierten Systemen und ihren Benutzern. Jedes wissensbasierte System benötigt Erklärungen, um ungenaue Domänen sowie komplexe Mechanismen genauer darzustellen.
Sie werden ebenfalls gebraucht, um den Entscheidungsprozess des Benutzers, ob die Empfehlung des Systems korrekt ist oder nicht, zu unterstützen. Entscheidungsträger müssen dem System und ihren Empfehlungen vertrauen. Falls der Prozess beziehungsweise die Empfehlung nicht verstanden wird, ist es unwahrscheinlich, dass diese vom Benutzer akzeptiert wird. Somit spielen Erklärungen eine entscheidende Rolle für die Akzeptanz wissensbasierter Systeme.
Diese Diplomarbeit präsentiert EM4J, einen Werkzeugkasten zur Erstellung von Erklärungen für das Intelligent Geo-Risk Evaluation System (InGES). Zusätzlich unterstützt EM4J die Erstellung von wissensbasierten Systemen. Es wird vom Institut für Geodäsie und Geophysik, sowie der Arbeitsgruppe Wissensbasierte Systeme (Institut für Informationssysteme) der Technischen Universität Wien, dem Institut für digitale Bildverarbeitung am Joanneum Research und dem Institut für Geologie an der Universität Bern, Schweiz, entwickelt. Die Aufgabe dieses Systems ist es, Risikostufen für untersuchte Landschaftsgebiete abzuschätzen. Da InGES in einer Umgebung arbeitet, die sich durch sehr hohe Kosten für Fehlentscheidungen auszeichnet, war es das Ziel, eine technische transparente prozess-orientierte Anwendung zu implementieren. Diese soll den Schlussfolgerungsmechanismus genau beleuchten. Dieser Ansatz soll dem Benutzer die Entscheidung, ob eine Empfehlung von InGES angenommen oder abgelehnt werden soll, erleichtern.
EM4J basiert auf der Theorie von Wick et al. Es ist eine prozess- sowie verifizierungsorientierte Erklärungskomponente entstanden, die eine sehr hohe Kopplung zum Folgerungsmechanismus hat. Somit ist EM4J auf eine hochqualifizierte Benutzergruppe zugeschnitten. Die Entwicklung selbst war geteilt in vier Phasen: die Implementierungs-, die Test-, die Integrations- und die Evaluationsphase.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Explanations are an essential part of everybody's life. People continuously give and receive them in order to process information which in some cases would not be apprehend to them. Similarly, explanations are also a crucial part within the communication between Knowledge Based Systems[1] (KBSs) and users. Every KBS needs to give explanations because of imprecise domains and the use of complex algorithms, heuristics, etc. [21]. Explanations are also needed to support the user in its decision process whether the recommendation is valid or not. Decision makers have to trust this recommendation as well as the system itself. If they do not understand the process as well as the conclusion it is unlikely that they will accept it. Therefore explanations play a vital role in the acceptance of KBSs.

1.1 Motivation

The focus of this work lies on rule-based systems (RBSs) which is a subcategory of KBSs. The rule-based approach is especially suitable for complex problems with developers and later on users possessing expert knowledge about the respective problem domain. The advantages of RBSs are:

- Rules are independent from each other. This feature eases the effort used for maintenance.
- This approach is easy to understand. The syntax of the rules is often very natural directly reflecting expert's knowledge.
- The inference engine is independent from the rules. This feature also supports maintenance.
- The structure of the rules supports the conversion into natural language.
- Rules can serve as explanations. Why and how questions can be answered fairly easily.

[1]Especially in older literature Expert Systems (ESs) often also refer to such systems

- The possibility to simultaneously process possible solution candidates along the way to generate the final solution.

The decision making of RBSs is strictly based on rules. Each rule defines preconditions and its corresponding actions.

IF *(conditions)* THEN *(actions)*

Rules have the form of conditions that are also called right hand side (RHS) and corresponding actions that are also called left hand side (LHS) of the rule. The principle for executing these actions is that the preconditions must be true. However this is not the exclusive prerequisite.

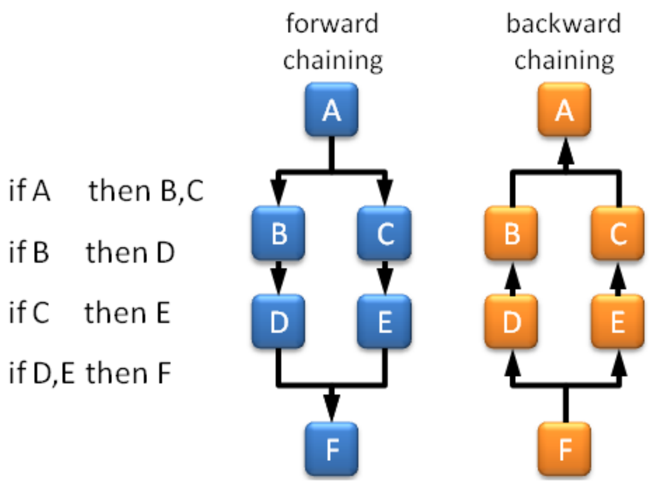


Figure 1.1: Forward and backward chaining principle

The basic RBS inference engine mechanisms respectively approaches are data driven (forward chaining), goal driven (backward chaining) or mixtures of them. Forward chaining starts with the facts out of the factbase, derives further information with the help of the rules out of the rulebase until the inference engine concludes in a goal. Backward chaining on the other hand starts with a certain goal and works his way back to the initial facts. The principle of the forward and backward chaining is presented in Figure 1.1. In the following we describe the common basic process steps of these approaches:

1. Each rule in the rulebase is matched against the factbase. In other words each LHS is matched against the facts present in the factbase.

2. Each rule that has its conditions satisfied is put in the conflict set. In most implementations this conflict set is represented as an ordered list.

3. Out of this conflict set one rule is chosen for execution. This means that not all satisfied rules actions are automatically performed. The selection process is called conflict resolu-

tion and depends on the implementation. The principle process is presented in Figure 1.2. There exist several rule selection strategies. Three of them are now listed below:

- **Degree of specialisation:** The most specific rule is chosen. In other words the rule with the conditions that restrict the factbase the most is taken.
- **Recency:** The rule that is satisfied with the most recently added facts is chosen.
- **Loop prevention:** Previously satisfied rules that are satisfied again with the same facts are ignored.

The performance of the inference engine is to a major degree influenced by the implementation of the conflict set resolution strategy.
If the conflict set is empty the inference engine stops.

4. The rule's actions are performed and the process starts all over again.

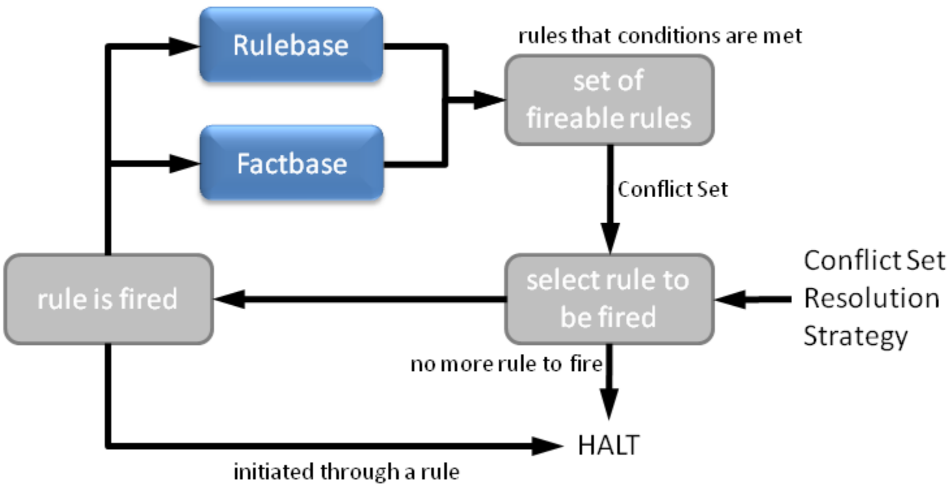


Figure 1.2: Conflict set resolution process

Only if a rule is chosen in the conflict resolution its actions are performed. The term used for the execution is that the rule is fired. To put it simple the computed conclusion of the RBS (not the explanation system) is found through concatenation of rule firings. This is performed until no more rule can be fired or some final defined state is reached (as already mentioned in the process description above).
The resulting concatenation of rules respectively rule trace constitutes the foundation for the generation of explanations in RBSs. In principal the rule trace contains all the information essential for the generation of explanations answering why and how questions. Why a certain conclusion was found respectively how it was generated. These two questions build the main category of explanations asked from RBSs since they are able to solidify the confidence towards the system. However a basic answer to them is often not satisfying enough to the user [11, 12]. Features like the overall reasoning strategy, the appropriateness of rules for a given problem,

basic domain principles, etc. can not be covered by the information contained in the rule trace. The question arises why we concentrated on the rule trace as a basis for EM4J's explanations. The driving factor behind this choice is simplicity. The decisions made by the system are apparent in the rule trace. They are represented by the fired rules which mirror the process from the initial state to the final computed recommendation.
The primary user group of out explanation systems are experts. On the one hand domain experts (i.e., geologists) that have a sound understanding of the principles the RBS investigates. On the other hand software developers that have a high expertise in the field of developing RBS.
In our case the disadvantages of the decision tree approach [11, 12] are not of that importance since the expertise of EM4J's users is very high. In other words these users possess deep knowledge about the domain. This eases the burden that weighs on the explanation facility. Two concrete issues are:

- The explanation facility does not necessarily need the capability to explain the appropriateness of rules
- The explanation system does not need the capability to explain basic domain principles.

The first is mitigated through the fact that the rules are developed by some of the users themselves. Therefore the users already know the appropriateness of the rules.
The second is solved through the expertise of the user group. The users are mostly domain experts consequently they possess a deep understanding of the problems at hand as well as the domain and its principles, basic rules, etc.
However one major problem is still persistent. Rule traces possesses a high percentage of "opacity" which means there is a lot of non-relevant information present. To clarify this issue further consider the following example:

```
==> Activation: MAIN::make_final_risk_class :  f-266,
==> Activation: MAIN::make_final_risk_class :  f-267,
==> Activation: MAIN::make_final_risk_class :  f-268,
==> Activation: MAIN::make_final_risk_class :  f-269,
==> Activation: MAIN::make_final_risk_class :  f-270,
==> Activation: MAIN::make_final_risk_class :  f-271,
==> Activation: MAIN::make_final_risk_class :  f-272,
==> Activation: MAIN::make_final_risk_class :  f-273,
==> Activation: MAIN::make_final_risk_class :  f-274,
==> Activation: MAIN::make_final_risk_class :  f-275,
==> Activation: MAIN::make_final_risk_class :  f-276,
==> Activation: MAIN::make_final_risk_class :  f-277,
==> Activation: MAIN::make_final_risk_class :  f-278,
==> Activation: MAIN::make_final_risk_class :  f-279,
==> Activation: MAIN::make_final_risk_class :  f-280,
==> Activation: MAIN::make_final_risk_class :  f-281,
==> Activation: MAIN::make_final_risk_class :  f-282,
==> Activation: MAIN::make_final_risk_class :  f-283,
==> Activation: MAIN::make_final_risk_class :  f-284,
==> Activation: MAIN::make_final_risk_class :  f-285,
MAIN::make_final_risk_class: =1=1+1+2+t
MAIN::final_riskpot_high_a: =1+1+1=1+1+2=1+1+2+t
MAIN::final_riskpot_veryhigh_a: =1+1+1=1=1+2+t
MAIN::final_riskpot_veryhigh_b: =1+1+1=1=1+2=1=1+2+t
MAIN::xxxx: =1+1+1=1+1+1+2+t
FIRE 1 MAIN::make_tendenz_class f-285,
 ==> f-286 (MAIN::tendenz (date 20100110) (+tendenz NN) (...
<== Activation: MAIN::make_tendenz_class :  f-262,
<== Activation: MAIN::make_tendenz_class :  f-263,
<== Activation: MAIN::make_tendenz_class :  f-264,
<== Activation: MAIN::make_tendenz_class :  f-265,
<== Activation: MAIN::make_tendenz_class :  f-266,
<== Activation: MAIN::make_tendenz_class :  f-267,
<== Activation: MAIN::make_tendenz_class :  f-268,
<== Activation: MAIN::make_tendenz_class :  f-269,
<== Activation: MAIN::make_tendenz_class :  f-270,
<== Activation: MAIN::make_tendenz_class :  f-271,
<== Activation: MAIN::make_tendenz_class :  f-272,
<== Activation: MAIN::make_tendenz_class :  f-273,
<== Activation: MAIN::make_tendenz_class :  f-274,
<== Activation: MAIN::make_tendenz_class :  f-275,
<== Activation: MAIN::make_tendenz_class :  f-276,
<== Activation: MAIN::make_tendenz_class :  f-277,
<== Activation: MAIN::make_tendenz_class :  f-278,
<== Activation: MAIN::make_tendenz_class :  f-279,
<== Activation: MAIN::make_tendenz_class :  f-280,
<== Activation: MAIN::make_tendenz_class :  f-281,
<== Activation: MAIN::make_tendenz_class :  f-282,
<== Activation: MAIN::make_tendenz_class :  f-283,
<== Activation: MAIN::make_tendenz_class :  f-284,
FIRE 2 MAIN::make_final_risk_class f-285,
 ==> f-287 (MAIN::FINAL_RISK (date NN) (final_riskpot NN) (fi...
==> Activation: MAIN::final_riskpot_veryhigh_a :  f-0, f-287
FIRE 3 MAIN::make_final_risk_class f-284,
```

This is an excerpt of a rule trace going over 107 pages from a RBS which fired 1906 rules while computing the final recommendation. The syntax of such an output is based on the inference engine and differs from engine to engine. There is a lot of internal information listed. This often causes problems for users because they are not familiar with it. This is especially true if the user does not have enough experience with rule engines and their working method. In most cases the essential information are the rule firings and the corresponding changes to the factbase. They show the system's course of action to compute the recommendation. The syntax of the fired

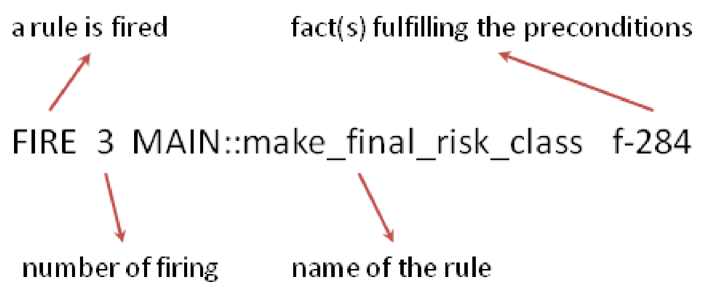

a rule is fired
fact(s) fulfilling the preconditions
FIRE 3 MAIN::make_final_risk_class f-284
number of firing
name of the rule

Figure 1.3: Example of a fired rule

rules of this excerpt is shown in Figure 1.3. The rules fired are marked with the letters "`FIRE`" at the beginning of the line. This is followed by the continuous number of the firing, the name of rule and at last the identifiers of the fact(s) causing this event.
The problem with this notation is that there is too much additional information shown to the user in a convenient way in order to convey the essential steps done by the system. Therefore a computer-aided enrichment process extracting the essential information has to be performed. Explanation modules like EM4J automatically perform this extraction. Additional effort is needed to develop the explanation facility. However the maintainability as well as usability of the whole system is significantly increased. The user is only confronted with the important parts of the trace which helps him to understand why and how the system computed the recommendation. In other words the opaque trace dump is filtered, structured and graphically enriched.
For completeness purposes we also mention two additional approaches which have some huge limitations.

1. **Manual Processing:** Manual processing performed by the user is often painstaking detail work. Consider going through 107 pages of rule trace extracting the decisions taken by the system. Especially huge systems with hundreds of rules are not manageable by humans. The processing of the rule trace is just too resource consuming and therefore some machine processing is absolutely needed.

2. **Semi-automatic Approach:** As a remedy the user can write additional information into the rules which ease the manual processing of the rule trace. This is often done via putting information to the output mechanism of the rule engine if a specific rule is fired. To stick with the trace example before an sample would be "The rule `MAIN::make_final_risk_class` is fired because of the following facts…". However this is just a transformation from one trace syntax into another. The subsequent processing still has to be done manually by the user as in the first bullet point. Although the information is a little more compressed than before it is still hard to convey the systems behavior.
   A disadvantage of this approach is that the adaption to rules can cause a huge effort during the development of the system and can significantly decrease the maintainability of the system. This is due to the fact that every change in the explanation capability requires a redevelopment of the rulebase.

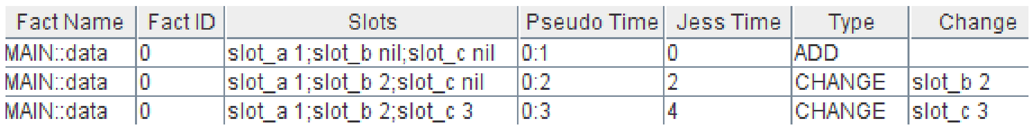
The main goal of EM4J is to ease this problem of processing the huge amount of data and consequently support the user's understanding. EM4J guides the user away from the plain trace dump through to a structured filtered graphically visualized decision tree. This tree reflects the reasoning of the system in a compact and comparatively manageable representation. The basic components are the fired rules and the changes performed to the factbase. Both are represented through vertices. Edges connect them together based on the decisions taken by the RBS. In other words if a rule is fired it has a connection to first every fact leading to the firing and second every change in the factbase. Changes in the factbase are modifications in at least one value of a fact, the addition of a new fact to the factbase or the removal of an already existing fact from the factbase.
To give an illustration of how such decision trees look like and how they are built we are going to look at the following example. The RBS in focus has a factbase composed out of one fact:

```
(MAIN::data (slot_a 1) (slot_b nil) (slot_c:  nil)
```

The fact is composed out of the name followed by a list of (slot value) pairs. This fact has initially only the first slot set. The other two are empty (represented by `nil` in Jess). The rulebase on the other hand contains three rules:

```
MAIN::match-1_:  IF (data.slot_a=1) THEN (data.slot_b 2)
MAIN::match-2_:  IF (data.slot_b=2) THEN (data.slot_c 3)
MAIN::match-3_:  IF (data.slot_c=3) THEN ( print "END" )
```

The first rule's (`MAIN::match-1_`) condition is fulfilled if there exists a fact in the factbase of type `data` with the value of the slot `slot_a` equal to 1. The second rule's condition is fulfilled if the value of `slot_b` is equal to 2. The third rule's condition is fulfilled if value of `slot_c` is equal to 3. The action of the first rule leads to the slot `slot_b` of the fact used to fulfill the condition being changed to 2. The second rule sets the slot `slot_c` to 3 and the last rule has no factbase changing actions. However, it prints the word `END`.



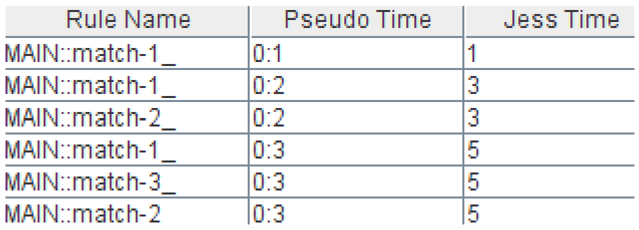
Fact Name	Fact ID	Slots	Pseudo Time	Jess Time	Type	Change
MAIN::data	0	slot_a 1;slot_b nil;slot_c nil	0:1	0	ADD	
MAIN::data	0	slot_a 1;slot_b 2;slot_c nil	0:2	2	CHANGE	slot_b 2
MAIN::data	0	slot_a 1;slot_b 2;slot_c 3	0:3	4	CHANGE	slot_c 3

Figure 1.4: Exemplary factbase history[2]

Figure 1.4 shows the complete trace of fact changes which is called the fact history. The first line represents the initial fact at the time it is added. Each and every following line represents this fact after each modification. That we are always dealing with the same fact can be seen in the "Fact ID" column. The entry here is always `0`. Every fact is equipped with a unique global timeline. The actual time stamp of each fact representation is shown in the "Pseudo Time" column. It shows the evolution of the fact as the system computes its recommendation. This

[2]Pseudo Time denotes the timestamp of the fact assigned by EM4J. Jess Time denotes a Jess internally assigned time stamp

time is composed out of the fact id followed by a colon and a serial number unique for this version of the fact.
Figure 1.5 shows the corresponding rule trace. With the help of the pseudo time column the moment each rule was fired can be reconstructed.

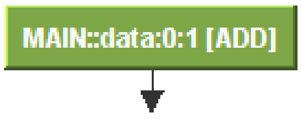


Rule Name	Pseudo Time	Jess Time
MAIN::match-1_	0:1	1
MAIN::match-1_	0:2	3
MAIN::match-2_	0:2	3
MAIN::match-1_	0:3	5
MAIN::match-3_	0:3	5
MAIN::match-2	0:3	5

Figure 1.5: Exemplary rule trace[3]

Generating the decision tree is then based on this rule trace as well as on the fact history. The exact generation method of our example RBS is now listed below:

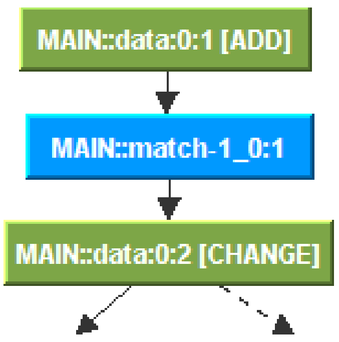
1. One fact is present in the factbase. It builds the root of the decision tree or the source of reasoning. This node visualizes the first line in Figure 1.4. The name of the node is

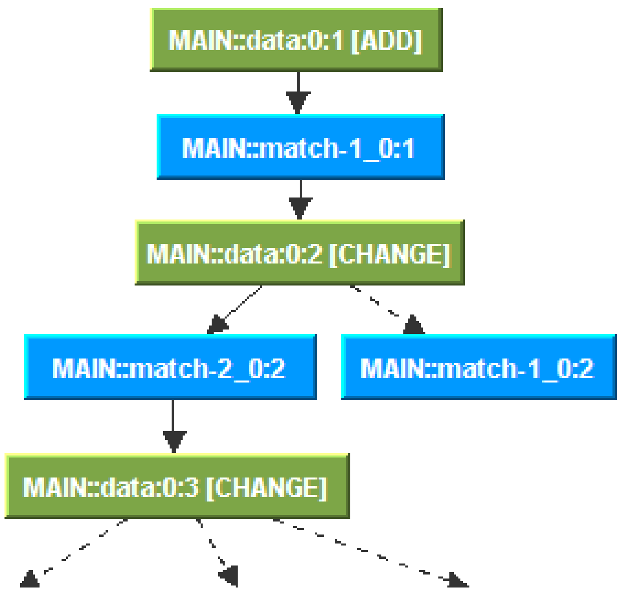
   composed of the name of the Jess module the fact is placed, the name of the fact underlying template or type as we called it before, the unique identifier and the type of modification happened to the factbase. Each of these parts are separated by a colon.
   The fact itself has three slots with the first one `slot_a` filled with `1`. These characteristics are graphically presented to the user via tooltips attached to the node and the fact history table in Figure 1.4.

2. This fact with the timestamp `0:1` (see the first line in Figure 1.4) fulfills the condition of rule `MAIN::match-1_` which is then fired. The name of this rule node is composed of the Jess module the rule is placed, the name of the rule and the timestamps of the facts that fulfill the conditions of the rule. They are again separated by colons. This rule also possesses the timestamp `0:1` (first line in Figure 1.5).
   The action of this rule leads to the slot `slot_b` being set to `2`.

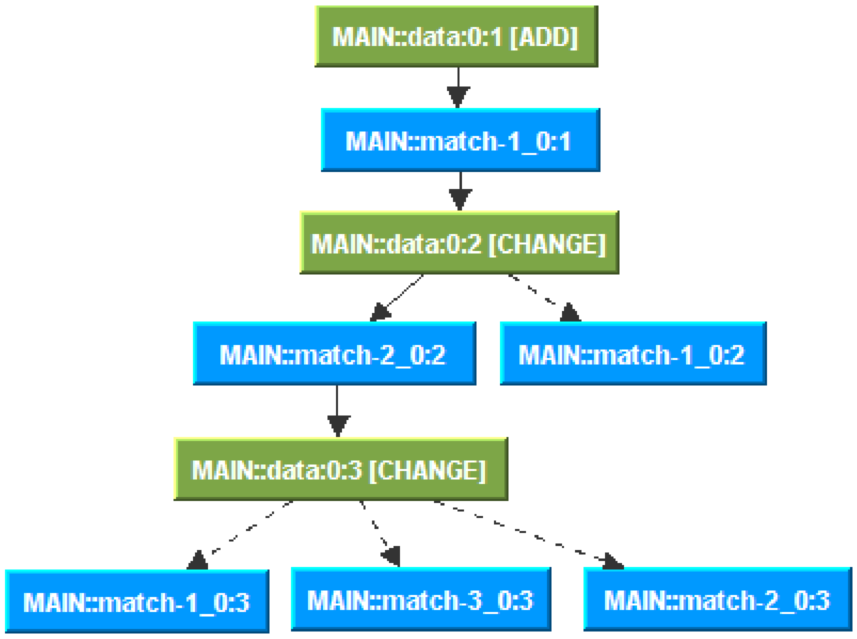
[3]Pseudo Time denotes the timestamp of the fact assigned by EM4J. Jess Time denotes a Jess internally assigned time stamp



3. The condition of rule `match-2_` is fulfilled and it is fired. Rule `main-1_`'s condition is still met. It survived the conflict set resolution and is fired again. However, its action has no visible affect to the factbase. Slot `slot_b` is already set to `2`. In order to clearly illustrate this circumstance to the user the arrow leading to node `MAIN::main-1_0:2` is dashed in the decision tree.
   The action of `match-2_` sets the slot `slot_c` to `3`.



4. Last but not least all rules are fired. Rule `match-1_`'s and `match-2_`'s condition are met again. They are fired but have no visible affect to the factbase since their actions are already present. Rule `match-3_` condition is now also fulfilled. The rule is fired, the output `END` is printed and the system terminates.



This sample highlights the decision tree generation process of EM4J (Explanation Module toolbox for Jess). It is a toolbox to generate explanations for Jess (Java Expert Shell System). EM4J is an add-on to the Intelligent Geo-Risk Evaluation System (InGES), developed by the Institute of Geodesy and Geophysics and the Knowledge Based Systems Group (Institute of Information Systems) at the Vienna University of Technology, the Institute of Digital Image Processing at the Joanneum Research and the Institute of Geological Sciences at the University of Bern, Switzerland. The KBSs incorporated into InGES which perform varying tasks are developed in the Jess Rule Language. The driving factor behind the development of this toolbox was that there exist no commercial explanation facilities for the rule engine Jess.
As already mentioned the user group of the overall system are experts. Therefore the objective of EM4J is to deliver explanations in form of a highly transparent technical view inside the actual KBS running on the Jess rule engine. The core features of EM4J are:

1. A tool to investigate the decision tree. This tree is constructed out of the rule trace in connection with the factbase and its changes, characteristics and attributes. We already saw an example of this process.

2. A tool to investigate static dependencies between rules.

3. A tool to test the behavior of the system under differing initial configurations.

The capability to investigate the course of computed recommendations over all possible inputs is a valuable asset to evaluate the overall performance as well as the correctness of the system. These possible inputs are restricted by the definition of the knowledge representation underlying the KBS. This definition determines among others the feasible input values, possible states respectively final recommendations of the system.

The critical factor determining the acceptance of InGES is its correctness. EM4J is developed to explain the recommendations computed by InGES. InGES is a risk evaluation system for geological accidents such as landslides or rockfalls. Each geological site (e.g., hillside) is categorized by human experts via determining its characteristics (e.g., slope of the hillside). This process implies that errors, variations, etc. are possibly incorporated. This fact alone increases the anyway high complexity of the risk factor calculation performed by InGES. Consequently a strong supporting tool like EM4J is substantial to increase the confidence into InGES.

1.2 Related Work

We are now briefly going to look at some related work in the research field of KBSs.

1.2.1 Approach according to Swartout et al.

Swartout et al. [24] defined five requirements for explanation capabilities of KBSs. These requirements are called desiderata. The first desideratum focuses on the procedure of creating explanations, the second and third one impose requirements on explanations, the fourth and fifth discuss impediments that are generated through including an explanation facility into a KBS. For more information we refer to Section 2.1.2.1.

1.2.2 The approach according to Sormo et al.

Sormo et al. [23] stated that explanation techniques are not only limited to one category that are based on defined goals. Explanations are rather capable of being in multiple categories with varying importance. The defined categories are transparency, justification, relevance, conceptualization and learning. For more information we refer to Section 2.1.2.2.

1.2.3 Approach according to Wick et al.

Wick et al. [26] viewed KBSs and their explanations modules as two separate entities either having their own knowledge, problem solving procedure, etc. They presented a view which in fact is very similar to the one proposed by Swartout et al., although the structure and terminology is quite different. For more information we refer to Section 2.1.2.3.

1.2.4 QUE - Quering the expert

This paper presents an expert system explanation module (QUE) [15] which answers “why not” types of questions. They address specifically these types of questions within rule- and object-based reasoning procedures. “Why not” questions are answered by showing the explicit object(s)/rule(s) which deleted or changed data according to the question. If no object(s)/rule(s) ever existed the question is matched and the object(s)/rule(s) with the most similarity are chosen or if an object(s)/rule(s) exist(s) that could have matched the question then why it was not fired is represented.

1.2.5 Explanations Using Ripple-Down Rules

This paper proposed an alternative to the common explanation concepts by using so called ripple-down rules [21]. The ripple-down rules paradigm delivers a new perspective on the possibilities to generate explanations. The uniqueness of this approach comes from the combination of three aspects (knowledge representation exception structure, case usage for knowledge acquisition and retrospective knowledge analysis) which are often used one by one or as a combination of two of them but never all three together.

1.3 The Structure of the Thesis

This thesis is structured as follows. In Chapter 2 we are going to tackle the problem of the definition of good explanation facilities and summarize approaches from the literature. In the course of this chapter we also highlight the background to our design decisions. We take a look at the nature of explanation requests and especially at the effects of the user's expertise on the desired characteristics of explanations. These aspects are not particularly related to RBSs they are to a greater degree directed at KBSs in general. Chapter 3 introduces the architecture of the toolbox by describing the principles, algorithms and correlations behind all the tools. In Chapter 4 we look at the system from the user's point of view by displaying the functions and its handling contained in EM4J. In Chapter 5 finally a performance evaluation of EM4J is presented. We assess the performance by stretching the system to its limit. This is accomplished through the computation of test cases.

CHAPTER 2

Background

Explanations play a big part in the daily business of mankind. We give and receive them in all different kind of situations. The goal of this chapter is to give an overview of different approaches that surfaced in the literature during the last 30 years. Additionally we look at how they can be used to generate explanations in various forms and content. Having this basis we then go on to depict the effects of the user's expertise on requested explanations to round down the picture. This background will help later on to convey why EM4J is designed the way it is.

2.1 Evolution of Explanation Goals and Types

2.1.1 First-Generation Knowledge Based Systems

Generally speaking first-generation KBSs[1] give four rudimentary types of explanations ([23]):

1. **Reasoning Trace:** Explanations are generated out of the reasoning process used in the KBS to generate the solution (how and why the system reaches a certain recommendation).

2. **Justification:** These are explanations which are given in the form of more detailed background respectively domain knowledge. The idea behind this additional information is to increase the trust of the user in the reasoning process as a whole or just a single reasoning step.

3. **Strategic:** Explanations concerning the global strategy of the system are presented.

4. **Terminological:** These explanations try to illustrate domain terms, definitions and concepts in a way so that the user gains a better understanding of the system.

[1]The term first-generation KBSs refers to approaches and systems developed during the 80s of the previous century [21].

In principle these four types are qualified to satisfy the users need for explanation. Unfortunately these explanation types did not take into account special characteristics such as why users request explanations [13]. For further reading on first-generation KBSs and their explanations we refer to Chandrasekaran et al. [4]. This paper is one of the first within the research community to explore the problem of explanations in detail. By presenting a decomposition of this issue they come to the conclusion that there exist three top-level components:

1. How the system is able to picture its activity to fulfill the information need of the user. The language and representation used in this particular aspect is major for accomplishing user satisfaction.

2. How the user's characteristics, e.g., goals, knowledge, preferences, are utilized to generate an explanation from the information gained in the first bullet point. In the center of attention are the level of detail, the terms used, etc. These characteristics should be carefully elected in consideration of the human being, enabling him to grasp what is presented.

3. How the user interface that visualizes the information is capable of doing so. Especially graphics, natural language insight and how to create it must be considered here.

These top-level components are then used to form three types of explanations:

1. **Type 1 - How:** This type of explanation binds the actions performed during the reasoning process to knowledge and information that characterizes them. In other words how for a specific task the pieces of the puzzle are put together. Knowledge from the factbase, input data, rules, reasoning decisions, etc., are put together to form the conclusion.

2. **Type 2 - Justification:** Often a deeper look into the factbase itself is needed. For example consider environments with high failure costs. In such environments the knowledge contained in the system must be investigable in order to eliminate potential errors.
   These explanations should facilitate the user to convey the principles underlying the KBS. The purpose of Type 2 explanations is exactly to fulfill this demand. These illustrations are called justifications because they warrant to the user why this information was important to get to the encountered conclusion respectively recommendation. Such in-depth knowledge can be obtained in three ways [4].

   a) The first way is via references to outside knowledge. This includes citing books, manuals or any other source.

   b) The second way is via statistical appreciation of values. Results of statistical analysis (e.g., if it rained for about two weeks, in 80% of the time a slope began sliding) are used to comply the users information need.

   c) The third way is via additional domain knowledge. Often rationales and basic principles are used to sound the structure and composition of the knowledge in the system. Most of the time this leads to meta-information not applied in the decision making process but nevertheless it is of high importance.

3. **Type 3 - Control Strategy:** With these explanations the design decisions made by knowledge engineers and domain experts are manifested to the user. Without this capability the strategic basement and reasoning philosophy is not present to the user. They are just indirectly incorporated into the KBS behaving that way.

The just cited approach discusses the problem area of explanations in greater detail. These added dimensions highlight the diversity of explanations but still do not take into account the user they work for and the context they are in. In some sense this extended approach is very similar to the four primitive explanation types mentioned at the beginning of this section. However this discrete classification of explanation types and goals delivers a deeper understanding of the complexities that come with explanations. This is just a first step but indeed a major one in order to obtain a comprehensive theory about explanations.

2.1.2 Second-Generation Knowledge Based Systems

The next generation, so called second-generation KBS[2], brought some more sophisticated approaches. In this work three major often cited approaches are going to be presented [23, 24, 26]. Each of them tackles the problem differently and therefore is worth a deeper look. Although the essence of explanation features, goals and requirements are nearly the same, they are compounded into more detailed structure. This constitutes on an even deeper understanding of the nature of explanations.

2.1.2.1 Approach according to Swartout et al.

Swartout et al. [24] defined five requirements for explanation capabilities of KBSs. These requirements are called desiderata and explained in detail in the following. The first desideratum focuses on the procedure of creating explanations, the second and third one impose requirements on explanations, the fourth and fifth discuss impediments that are generated through including an explanation facility into a KBS.

1. **Fidelity:** This goal describes the accuracy of the explanation's representation. It is crucial that the knowledge used for the explanation generation is the same the KBS is using for its reasoning process. This seems nothing special if the explanation generation and the reasoning are performed by the same system. However if these are two seperate systems or modules this circumstance does not hold necessarily.
   Considering the fact that inaccurate explanations are worsening the trust into the system as well as its acceptance by the user this requirement is even more important.

2. **Understandability:** This is a major requirement to every explanation as its main focus lies on the user and its perception. The aim of understandability is that explanations should be helpful to the user. The user should get a better perception of the KBS.
   The following shows a non-exhaustive lists of criterias that are all interwoven under the

[2]The term second-generation KBSs refers to approaches and systems developed from the 90s of the previous century on [21].

pretence of understandability. Each plays its part in defining the significance of the explanation.

- **Terminology:** The user and its knowledge must be taken into account during the definition procedure of the terms used within explanations. Either the terms are known and understood by the user or the system must define them in a way which is familiar to the user.
- **User Sensitivity:** The system needs to be aware of the user's knowledge, goals, preferences, and concerns during the generation of explanations.
- **Abstraction:** Different levels of abstraction should be part of the explanation capabilities. By this it is meant that the terminology used is conceived for different classes of users (knowledge engineers, design experts, novice users, etc.).
- **Summarization:** Not to mix up with abstraction the system should incorporate various levels of detail into its explanation module. In particular by increasing the portion of knowledge within the explanation, e.g., by adding in-depth domain knowledge, or a more fundamental representation of the reasoning process.
- **Perspectives:** The ability to serve the user with several perspectives on the knowledge contained in the explanation.
- **Linguistic Competence:** If explanations are generated in human language they should hold on to linguistic principles. Natural sounding and a smooth textflow are crucial to increase the acceptance of the system.
- **Feedback:** Users tend to ask further questions regarding parts of the explanation who are not fully understood. That is the reason why an explanation facility should be able to communicate with the user to furnish in-depth information.

3. **Sufficiency:** The knowledge needed for reasoning over a specific domain is often not enough to support explanation capabilities. Additional deep knowledge about the domain is required to be able to answer the users questions. A useful strategy to identify this demand is to reverse the picture. Looking from the explanation's point of view is a popular asset. Some others are listed below:

   - **System's behavior explanations:** Such explanations reflect on how the system found its recommendation. Additionally of peculiar interest is how different configurations of the knowledge present in the system affect the computed recommendations. The reasoning trace, the reasoning process itself and an event history gathered through the problem-solving procedure provide the basis for these illustrations.
   - **Justifications:** KBSs not only have to choose the correct/appropriate steps in order to solve a problem, they also have to be able to argue why these steps are chosen. Consequently the system that generates the explanations requires a comprehension of the design decisions underlying it.
   - **Preferences:** A KBS frequently has to choose between recommendations, strategies, etc., to solve an existing issue. Explanations of this sort have to clarify why a certain preference is picked.

- **Domain explanations:** The center of attention here is the problem domain not the system working in this field. The other types use information which is gained through the factbase and/or the problem-solving procedure. Here the user should gain assurance whether a system and its knowledge is suitable for a given problem or be fed with information to better grasp the domain.
- **Terminology definitions:** The meanings of terms used by the system should be presented in a coherent form to support the user.

4. **Low Construction Overhead:** Explanation facilities cause extra costs to the design and development of a KBS. Instead of avoiding such modules, to keep the costs low, two aspects should be taken into account. First to minimize the calculation overhead created by an explanation module. This can be achieved through an intelligent and elaborate design. Second by justifying the costs through the extraction of additional benefit from the explanation capabilities (e.g., additional information gained through domain principles demonstrated by the explanation module).

5. **Efficiency:** Since run-time behavior of computer systems is almost always crucial for people using it, the system's performance must not degrade.

Swartout et al. deliver one of the most enhanced perspectives on explanations often considered as the baseline for developing state-of-the-art explanation facilities. Not only the user, but also the context is taken into account. This represents a notably upgrade to the maxims developed in the early years of research. This approach can only be matched by the ones from Wick et al. and Sormo et al. [23, 26] that will be cited later on.

2.1.2.2 The approach according to Sormo et al.

The second approach that we want to highlight introduced an alternative classification of explanation goals [23]. They state that explanation techniques are not only limited to one category that are based on defined goals. Explanations are rather capable of being in multiple categories with varying importance.

1. **Transparency:** Transparency is gained with explanations that answer how the system found its recommendation. There exist two primary user groups. First knowledge engineers who are interested in this type of explanations because of the ability to review and/or debug the KBS in order to assure correctness which is crucial. Especially in the occasion of anomalies, inconsistencies or contradictions it is vital to ask why they occur. Second domain experts who may also be interested in verifying and clarifying the problem-solving process. Transparency is related to the already mentioned fidelity. However such transparent explanations often require a priori knowledge from the user. Especially technical or domain background knowledge is asked from the user.

2. **Justification:** Confidence in the recommendation of the KBS leads to better acceptance by the user. Application fields where the cost of failure is low a certain degree of simplification respectively abstraction from the problem-solving process can be adopted. The

purpose is to support as well as facilitate the understanding especially for novice users. High failure cost environments on the other hand absolutely require in-depth knowledge about the process in their explanation.

3. **Relevance:** Similar to justification, where the recommendation is validated, relevance discusses the strategy of the KBS. In detail the global recommendation finding strategy that was applied is examined. Here we have to mention that explanations in the form of reasoning traces are just able to illustrate the strategy in an implicit way. However no argumentation on why the strategy is appropriate, good or even the best one is performed.

4. **Conceptualization:** Concepts and their relationships are part of every KBS. Almost all application domains possess their own characteristics and forms. The user is possibly overwhelmed by their variety. Consequently another vital aspect of explanations is to explicitly deal with the vocabulary of the KBS.

5. **Learning:** There also exist educational application fields where a KBS can be of benefit. Such educational systems have as its main intent to propagate understanding of the domain to the user. The quality of the conclusions of such a system is not the primary subject of investigation. Instead the way how the solution was obtained is in the focus of explanations. Additionally these systems have to be aware of the differences in the way how problem-solving is done by humans and computers.

These five categories picture a more machine-oriented approach with the aspects of human users indirectly incorporated into the characteristics.

2.1.2.3 Approach according to Wick et al.

Wick et al. [26] viewed KBSs and their explanations modules as two separate entities either having their own knowledge, problem solving procedure, etc. They presented a view which in fact is very similar to the one proposed by Swartout et al., although the structure and terminology is quite different. The basis for this approach is the circumstance that human experts possess the natural talent to perform complex reasoning to solve a given problem. Additionally they will wrap this reasoning process into a story that presents the key facts in a human understandable form respective explanation. Human experts will seldomly give an explicit reasoning trace (except for instance discussing with a fellow expert) in order to explain their solution respectively recommendation. Conclusively the original line of reasoning can aberrate from the line of explanation. Generally speaking it is not just a rewriting of rules respectively reformulation of the reasoning process. On the other hand explanations may contain added information not available during or used in the reasoning. There is a decoupling between the reasoning process and the explanation generation prevalent. In this perspective of coupling respectively decoupling Wick et al. presents three major features of explanations:

1. **Goal:** The aim of an explanation can be:
    - **Verification:** Its purpose is to validate the knowledge of the KBS. This is achieved by displaying the knowledge of the system in a clear, appropriate, correct, plain and uncomplex way with the result that the user fully understands.

- **Duplication:** The primary purpose of this goal is to transfer the knowledge from the system to the user rather than to just prepare it in a comprehensible way. The aim is to communicate the knowledge respectively the reasoning of the system to the user with the intention to learn him/her the reuse under differing circumstances (e.g., system configurations, input, etc.). This type of goal is perhaps the most difficult to judge. What makes a good duplication explanation? The only measure covering this is to quantify the extend to which the user is able to emulate as well as simulate the systems problem-solving process.
- **Ratification:** The central point in this one is trust. The explanation should provide the user an understanding of the domain. This goal can easily be mixed up with duplication. However this type of explanations aim at the domain the system is deployed not the knowledge contained in the system. In other words the domain itself is explained without lying the focus on the system computing the recommendation. A good ratification explanation allows the user to judge the systems usability and therefore raises the comfort level towards the system.

2. **Audience:** The audience has a strong influence on the content of each given explanation. By reviewing the three major audiences proposed by Wick et al. we will see why this is true:

   - **Knowledge Engineer:** This type of user usually demands explanations that deliver a transparent view into the internal mechanisms of the system in order to evaluate it.
   - **Domain Expert:** In general this type of user relies on his/her domain expertise to evaluate exactly whether the information contained in the explanation is sufficient and presented in an accessible way.
   - **End-User:** This type of user first and foremost has the desire to gain a sound comprehension of the KBS and its advice. In this way the explanations should support the decision process.

3. **Focus:** Last but not least the focus is another key player in determining the tenor and the shape of the explanation. There are two main types:

   - **Process:** As its name implies the target here is to highlight the reasoning respective problem-solving process. Detailed information about how the recommendation respectively solution is found is represented.
   - **Solution:** Solution-oriented explanations on the other hand analyze the recommendation respectively the solution itself. In other words the target here is to screen the content and argue why it was generated.

A deeper look at these three features show that they are by no means independent from each other. Indeed they interact in a profound way to form explanations. Consider the following example. A domain expert is usually more interested in evaluating the domain knowledge build into the system. In other words he/she aims at two key characteristics. First at verifying the level

of detail and second at verifying the correctness of the information incorporated into the fact-base and reasoning logic. On the other hand end-users usually just want to use a system while not being bothered with such highly complex internal issues. It can be seen that these attitudes strongly influence the shape of the respective explanation request.
Wick et al. suggest that the verification- and process-oriented explanation results in a higher degree of coupling between the explanation module and the problem-solving mechanism. On the other hand the ratification- and solution-oriented explanation needs a higher degree of de-coupling of the KBS and the explanation facility.
As a side note Sormo et al. states that decoupling and fidelity are inversely proportional to each other. Fidelity is characterized by the fact that the knowledge used for reasoning is also used for generating the explanation. Consequently an increased decoupling results in a decreased fidelity. Human experts also tend to decouple the knowledge used during the reasoning process from the information wrapped into an explanation. Nevertheless they are still in most cases appreciated by the user.
After an closer examination we decided to base EM4J on the approach of Wick et al. The idea behind EM4J, its targeted user group and the rule-based approach perfectly match to the concept of Wick et al. The distinction respectively categorization into three focus areas in connection with the decoupling maxim goes hand in hand with the objective target of explaining the working method of RBSs. EM4J pursues a rather coupled, process- and verification-oriented approach.

2.1.2.4 Summary of the three Approaches

In conclusion the explanation goals of Wick et al. are defined from the perspective of human users whereas the approach of Sormo et al. is more user independent. In the approach of Wick et al. the user is recognized as a key factor that shapes the form and content of the explanation. Sormo et al. do not consider the user as a separate factor. Their approach leans on the character-istics, the form and the content that is provided by the explanation facility.
The explanations of Wick et al. are structured to a higher degree with regard to goal, user and focus. In contrast, the definitions of Sormo et al. possess a flat set-up more or less from a machine point of view. Wick et al. identified three key players with each of them influencing the explanation differently. These key categories are further divided into subcategories. The approach of Sormo et al. describes five aspects with the possibility of multiple categorization and no subcategories. However the user is not recognized directly as a category.
On the other hand Swartout et al. combines in some way these two approaches. Swartout et al. take the user into account (e.g.,Ünderstandabilitycategory) as well as purely technical aspects (e.g., LLow Construction Overheadcategory) It not only sees explanations from a user perspec-tive but also from a machine's point of view.
All this shows how difficult it is to clearly define the best explanation theory. Explanations have varying manifestations and are strongly dependent on environmental influences that characterize and shape the capabilities for the given purpose.
After highlighting the evolution, development and emergence of explanation goals we are going to look at the reasons for explanation requests and the usage of explanations in the next section. In this context the effects of the user's expertise are investigated with respect to what type of explanations the respective user group prefers.

2.2 Explanation Requests

Before we dig into the nature of explanation requests we want to dig a little deeper and highlight two theories that form the theoretical respectively philosophical background for the next section. These theories cover the tendency of the user to request explanations and should lead to a better understanding of how, when and why explanations are used.
The first approach is the **cognitive effort perspective** [18]. Human beings usually balance the pros (profit) and cons (costs) of taking a certain pathway. An empirical study was performed to verify that reducing the effort attracts larger interest than increasing the quality of the decision. This is a key reason to consider before designing an explanation facility. From this perspective, simplicity and accessibility should enjoy a high priority.
Gregor et al. [9, 10] then expanded this theory to the usage of explanations of KBSs. Their suggestion is that the use of explanations in KBS is affected by pro-con-trade-offs. In short the consequence is that the usage of explanations can be increased by improving the accessibility, usefulness, simplicity, etc., in other words, by increasing the benefits together with a reduction of the costs.
With this background we now come back to the nature of explanation requests and the effects that the expertise of users have on them. This basis further fosters the facts presented below.

2.2.1 Nature

According to Mao et al. [14] there exist three primary categories why users want an explanation to outline the systems recommendation. Table 2.1 shows the category at the top of each section and the two concrete shapings below. Generally speaking users that possess a strong consensus with the system's behavior are less likely to call for explanations. In conclusion the user asks more likely for explanations if he fails to generate reasonable understanding and/or explanations by himself [14].

2.2.2 The Effect of Expertise

The differences between expert and novice users are being illuminated in this section. The exposition is based on [14]. This analysis has an immense influence on our decision to build EM4J the way we did.
Solving problems is a human discipline which is a part of the every day life. Gathering an understanding of the problem and therefore creating a cognitive representation of it is crucial in order to solve the problem.
According to Mao et al. expert and novice users possess remarkable differences in the way their perception out of the given knowledge is filtered and formed. Furthermore their skills to handle problems of high complexity underlies great variations. The basis for this issue is composed out of the following three coherent dimensions [19]. First the quantity by means of known concepts, second the capability to organize and structure these known concepts to given problems and third to access them fast from human memory.
A mixture of these three dimensions results in experts leaning towards a more fundamentally

Categories	Details
Understanding	
Significance and/or Relevance of a Conclusion	*Why* the system came to this conclusion. The user perceives the recommendation unexpected or feels that an anomaly is persistent. Difficulties arise and the user wants to derive a justification for the outcome.
Reasoning Process	*How* the system came to this conclusion. The user understands the significance of the recommendation but does not fully comprehend the problem-solving process of the system.
Verifying	
Confirm or Compare	The user wants to compare his own explanation with the explanation from the system mostly in the case of unexpected outcomes. The result should be a pleased user with an understanding of how the system works.
Supporting Details	The user wants further information as support to facilitate his own explanation of the recommendation.
Resolving Contradictions	
Surprise	The user expects the system to behave in line with his opinion. However this is not the case. Therefore an explanation for the recommendation is wanted.
Disagreement	There are contradictions between the expected and the actual conclusion of the system. The user lacks information why this situation happened.

Table 2.1: Classification of explanation requests

sound understanding of the problem and its analysis, whereas novices most of the time stay on the surface not breaking through the problem's complexity. Additionally experts use goal-oriented strategies for problem-solving. Novices on the other hand frequently are delayed through searching for methods to detect and evaluate their actions and inferences. They rely on general problem-solving techniques rather than using simpler domain specific solution approaches used by expert users.
According to Mao et al. domain experts possess more expertise about the domain. This asset plays a huge factor in deriving further information in order to solve the problem. This may be needed to solve the problem in a convenient way. The derived information is normaly not directly given. It is furthermore hidden within the problem statement and the principles underly-

ing the problem. As already said before, novices are often not able to recognize this knowledge which furthermore leads to less competence in solving complex issues.
This now leads us to the classification of how explanations from KBSs are used [14]. Novice users have a greater need for explanations from KBSs (especially rule-based systems). This need comes from the missing knowledge and expertise that this user type has in order to understand and evaluate the conclusions of the system. Experts on the other hand possess two advantages. First they are able to identify the relevance of the solutions posted by KBSs and they have the ability to evaluate and explain them. Second experts tend to have more questions on implausible and inconsistent conclusions given by the system. The reason for this is that expert users normaly possess more skills to detect inconsistencies and question recommendations.
Experts are frequently able to find divergent ways to solve a problem. This can result in discrepancies between the strategic philosophy of experts and KBSs. In this case the reasoning trace can be a useful feature in order to clarify these discrepancies. Expert users request justifications to a lesser degree than novices. This is based on their overall higher capabilities respective knowledge, the problem solving practices known by them and the ability to detect the applicability of generic procedures on particular situations.
The capabilities, problem solving practices and the detection of the applicability of generic procedures on particular situations lead them to a lesser degree of justification need on the conclusions of KBSs than novices [14].
These facts are further underlined by a study based on tape recorded verbal protocols of the user-KBS interaction reported in [14]. The three reasons of explanation requests from Table 2.1 are utilized to categorize the results. The results say that nearly two thirds of the novice users wanted to increase their understanding of the recommendation. On the other hand experts devoted less than 50 percent of the requested explanations to increase their understanding. The verification results are also in line with the findings above. Experts used one third of the verification results in comparison to under 20 percent by the novices. The contradictions category shows just a slight difference (eleven percent by experts to eight percent by novices).
The results[3] from the usage analysis are merged with the most common explanation types, namely justification (*why*), reasoning-trace (*how*) and strategy. Only after this assemblage we perceive the full picture. Expertise has a vast influence on the demand for explanations and its characteristics. Justifications, *why*-type questions, and strategic explanations are the preferred ones by novice users (almost 60 percent of the requested explanations). Expert users favor *how*-type questions to conceive the reasoning process. Therefore reasoning traces are a precedented approach to fulfill this need (more than 50 percent of the requested explanations). It was further revealed that the overall amount of trace explanations, from novices and experts, was the highest (nearly 50 percent of the desired clarifications).
Reasoning traces have their flaws but they are still one of the most popular types of explanations. Especially skilled users and experts esteem this type of explanation because of its deep insight into the system namely the reasoning process and mechanisms leading to the recommendations.

[3] A detailed table of the results can be found in [14]

2.3 Alignment of EM4J's Design to Wick et al.

In the following we will see how the ideas of the approach in [26] are implemented and how this approach was used to shape the capabilities of EM4J.

2.3.1 Audience

Through the design of EM4J we made the commitment towards domain experts and knowledge engineers. According to the theory, a knowledge engineer requests transparency. It is gained with a look at the core mechanism of the rule engine. In detail this is accomplished by the following three aspects. First the decision tree and its features, second the tables that contain additional information about facts respectively rules and third by the ability to automatically investigate the differences in the behavior of the system under altering configurations. Especially this last asset is important for domain experts. The theory implies that the domain expert's goal is to verify the knowledge of the system. Investigating the behavior of the system supports this goal because such automated tests help to discover discrepancies and/or inconsistencies. This can possibly cause incorrect and/or erroneous computed recommendations. Considering high failure cost environments this circumstance is absolutely inacceptable and therefore has to be avoided.

2.3.2 Focus

The tight coherences between the audience and the focus guided us towards process-oriented explanations. The classic question of how a recommendation respectively solution was found is typically answered with decision trees respectively rule traces. This circumstance leads us towards providing these features in EM4J.
Additionally the capability to test the system under varying circumstances is added. This feature also fulfills our user groups desire to investigate the correctness of the system. In this context it must be considered that the effects of expertise regarding explanations (Section 2.2.2) also strongly influenced EM4J's design.

2.3.3 Goal

The main focus is on verification and ratification. The former is clear because of the explanation characteristics enjoined by the audience and the focus. In turn the focus is influenced by the audience. Ratification is according to Wick et al. unusual in connection with our audience and focus. However since EM4J works in a high failure cost environment it is essential to keep this goal in mind. We will later on explain this environment embeddedness.
Verification of the KBS's knowledge is aided through the decision tree, the rule dependency graph and especially the behavior test capability. Regarding this feature it is worth mentioning that the ability to test the KBS's behavior under differing circumstances represents a highly capable feature to verify the knowledge and mechanics contained in the KBS.
Ratification is complied with the same rudiments as before. First our intends to fulfill the needs of the verification goal serve this one as well. The critical factor here among others is the fact that

by verifying the correctness of the system the level of confidence will be raised. Additionally the effects of user's higher expertise levels hold a great deal in increasing the credit given to the system through our reasoning-oriented respectively process-oriented explanations.
Finally, we want to emphasize the point that checking the behavior of the system through the test case feature is essential to all three goals. It is able to:

1. help to check the correctness of the system (verification goal), e.g., by showing inconsistencies in the behavior during the run of special cases,
2. help the user to get a sound understanding of the system in terms of the strategic direction, the applied reasoning technique, etc. (duplication goal),
3. help the user to get a sound understanding of the domain knowledge contained in the system (duplication goal),
4. via the first three items to increase the confidence in the system (ratification goal).

In summary the tools in its entirety are able to create a condition of satisfaction for the user. EM4J does not provide one single right explanation. Instead a conglomeration of knowledge about the system is provided with the help of different investigation tools. The aim of them is to support the user in finding his own specific explanation why and how the system computed a certain explanation.

CHAPTER 3

EM4J - The Architecture

The Explanation Module toolbox for Jess (EM4J) is a toolbox designed to support the user to verify the correctness of the computed recommendations given by the Intelligent Geo-Risk Evaluation System (InGES). EM4J is more than just an explanation facility. In fact it also possesses the rule engine Jess. This architectural aspect was necessary due to the fact that reasoning data has to be gathered during run-time which forced us to incorporate the reasoning engine into the toolbox. Frequently explanation modules work with data provided by the reasoning process after termination. EM4J on the other hand gathers the relevant data on its own during run-time.
The textual debugging output provided by the rule engine Jess can be used for explanation purposes. However it is not potent enough to support all of EM4J's needs. We already discussed this issue in Section 1.1. This debugging output consists of a chronological list of rule activations, rule firings, changes to the factbase, etc. Conveying the decisions performed by the system based on this list is hard to accomplish. Generating a decision tree out of this trace is even harder to perform. It is just a static textual postscript of the work done inside Jess. Solely relying on this postscript it is not easy to gather additional information such as rule dependencies. For example these dependencies rely on the actual conditions leading to the firing and the consequently performed actions. This circumstance led to EM4J's information collection mechanism. The aim of this collection mechanism is to ease the effort needed to develop EM4J. Later on in this chapter we will see this mechanism in detail.
Wick et al. proposed that a process, verification-oriented explanation generated for knowledge engineers and domain experts requires a tight coupling of the explanation facility and the rule engine. EM4J adheres to this principle through:

- **Tight coupling:** The tight coupling between the inference process and the explanation facility is ensured since the rule engine is incorporated as a tool in the toolbox EM4J.

- **Process Orientation:** EM4J is process-oriented because of the incorporation of decision trees and rule dependency analysis into the toolbox. The decision tree mainly explains how the system came to its computed conclusion. In other words the course of rule firings and the corresponding history of factbase changes are graphically presented. The rule dependency analysis mainly explains why the system displays a certain behaver. It answers questions like why different rules block each other, what are the similarities between rules, etc. With the help of this feature the user gains knowledge about the behavior of the rules or in other words the strategy of the KBS.

- **Verification Orientation:** The verification orientation can be argued with the same aspects as with the process orientation. The decision tree and the rule dependency analysis are tools to verify the computed conclusion of the KBS because they highlight the knowledge used during the reasoning.

- **User group:** EM4J is designed to facilitate domain experts, in this case geologists, and knowledge engineers also geologists and software developers.

3.1 Structure of EM4J

We present the core features and mechanisms of EM4J. The toolbox consists of three distinct but still coherent subsystems. Figure 3.1 presents the detailed architecture of EM4J including the components as well as their interactions. These subsystems are described in detail:

- **Rule Engine:** This tool accommodates the rule engine that represents the core reasoning tool. It has the ability to run every KBS implemented in Jess. Beside computing recommendations this rule engine has the purpose to collect data about the KBS with the following procedure. A KBS is loaded into the Rule Engine which executes it, collects the raw data to provide them to the other tools and sends this data to the Knowledge Base Archive Manager. A more detailed presentation is given in Section 3.3.

- **Archive Manager:** The archive manager deals with the data collected by one of the rule engine instances. This data includes the rules fired, the history of fact changes, basic data about the executed KBS, etc. All other parts of the toolbox access these central data stores to get the information in order to perform their evaluations. There exist two different types in this system:

  - **Knowledge Base Archive Manager:** Manages the data in archives collected by the Rule Engine. These archives contain the rules fired, facts changed, etc. of a KBS that is under investigation.
  - **Test Case Archive Manager:** Manages the data in archives collected during the execution of one of the two test case types.

- **EM4J:** This is the actual part of the toolbox responsible for the explanation of why and how a KBS computed a certain recommendation respectively conclusion. The user chooses archives (via the Knowledge Base Archive Manager respectively Test Case Archive Manager) that represent the data collected during the executions of a KBS or a test case. They can then be investigated with the following tools.

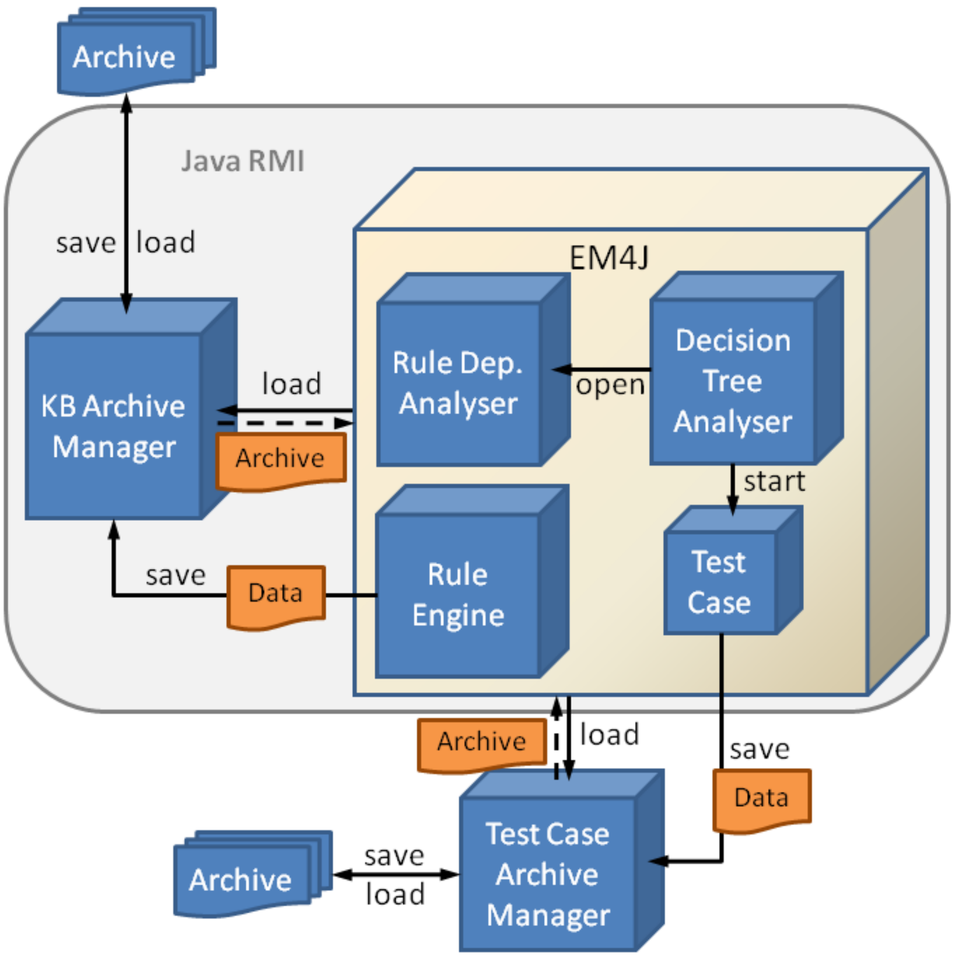


Figure 3.1: EM4J's detailed architecture

– **Decision Tree Analyser:** This tool facilitates the user in conveying the working method of the rule engine Jess. The core procedure is to build a decision tree out of the raw data. Additionally it can trigger the Rule Dependency Analyser as well as test cases. A more detailed presentation is given in Section 3.5.

– **Test Case:** Test cases facilitate the user to investigate the behavior of the KBS with varying inputs. In other words a KBS that was executed before, and its data stored in an archive, can also be investigated with other initial configurations. These alternations are then executed to check the alterations in the recommendation respectively conclusion. For that reason the test case functionality possesses an instance of the Rule Engine. There exist two types of test cases in the toolbox:

  ∗ **Simple Test Case:** They are used to test and investigate a single configuration. Visualization of the results is performed with an instance of the Decision Tree Analyser. This tool additionally possesses the capability to generate printable reports (e.g., in PDF).

∗ **Advanced Test Case:** This type is used to automatically test and investigate a range of configurations. Here, only a report is generated because visualizing every instance of the range configuration would go beyond the scope of any display. Figure 3.3 symbolizes this through several Simple Test Cases.

For more information about the capabilities and handling of test cases, we refer to Chapter 4.

– **Rule Dependency Analyser:** This tool possesses the capability to display to the user static dependencies between rules of a RBS implemented in Jess. It can either be started as a stand-alone application or via the Decision Tree Analyser. For more information we refer to Section 3.4.

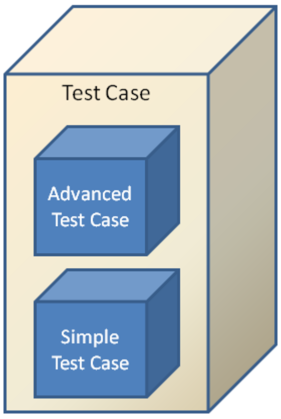


Figure 3.2: Test case architecture

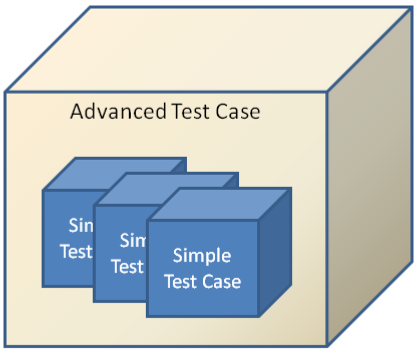


Figure 3.3: Advanced Test Case architecture

To add modularity and scalability to the system the Knowledge Base Archive Manager, the Rule Engine and EM4J communicate via Java RMI[1]. Thereby these sub-applications can be deployed over a network.
Next we want to present a description of the Intelligent Geo-Risk Evaluation System (InGES) before we explain the architecture of EM4J in more detail. EM4J is created to produce explanations for the KBS InGES.

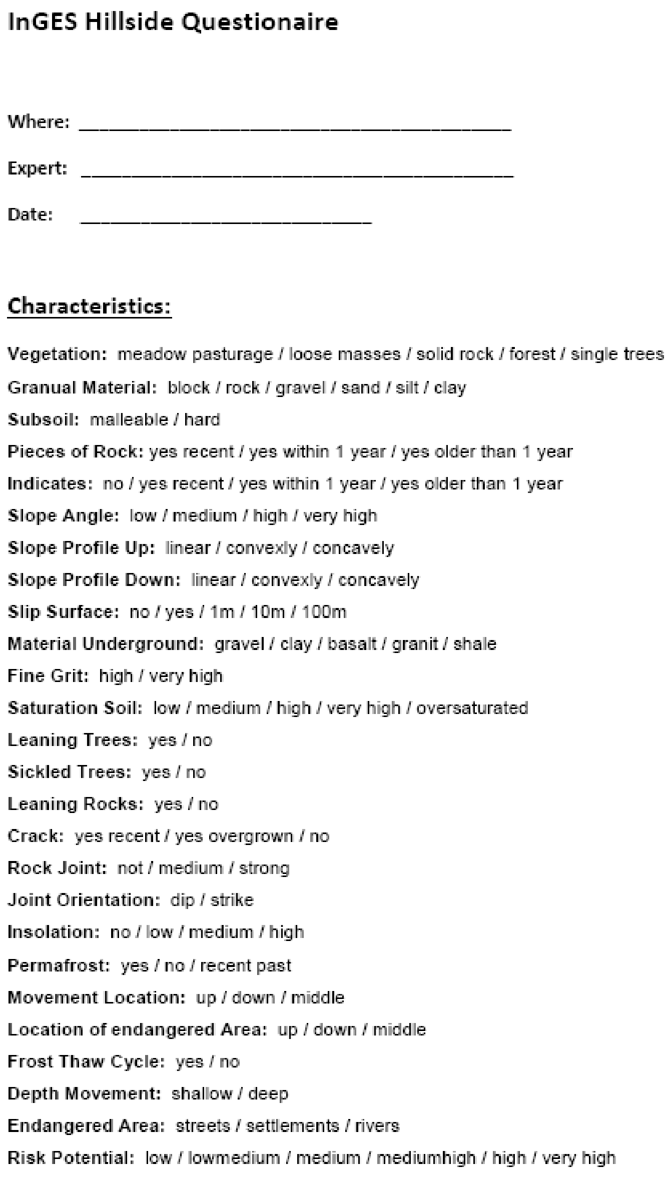
3.2 InGES

InGES [20, 25] is the product from an interdisciplinary research project called *i-MeaS*[2]. This project is a cooperation between the Institute of Geodesy and Geophysics and the Knowledge Based Systems Group (Institute of Information Systems) at the Vienna University of Technology, the Institute of Digital Image Processing at the Joanneum Research in Graz and the Institute of Geological Sciences at the University of Bern, Switzerland.
The main motivation for the development of InGES is the increasing number of geological incidents such as rockfalls or landslides. Heavy rain is a driving cause for these events. In connection with the prediction of many global climate change scenarios that weather such as heavy rain have an increased probability there exists a need for reliable geological monitoring tools. Although there exist some tools in the literature [1–3, 8, 22] there is still need for improvement. For example newer research topics such as deformation interpretation [20] are not covered.
The basis for such a geo-risk evaluation system is the capability to recognize, understand and lessen the risk of such incidents as well as reduce the consequences of them. These three critical characteristics can be achieved with a KBS. Such a system is capable to automatically interpret gathered geological information and identify the appropriate risk potential.
The core of this system is a KBS which is fed with geological information like the previous presented deformation vectors. In fact this represents the innovative aspect in this approach. The main purpose of the *i-MeaS* research is to develop a risk evaluation tool for geological changes (e.g., rockfalls, landslides, etc.). It possesses the ability to continuously process deformation information along with the capability for issuing alerts respectively computing risk levels.
For a more detailed description of the system we refer to [20, 25].

3.2.1 Risk Levels

The basic concept behind the identification of the appropriate risk level is composed of two concepts. The first is the "calculation" of risk factors. These factors are determined for critical variables that have a high probability of causing geological changes respectively accidents (e.g., landslides or rockfalls). Second a detailed exploration of the deformation has to be performed. The range of values for the risk factors is divided into six classes (low, low-medium, medium, medium-high, high, very-high). This scale is defined by geologists based on their expertise respectively experience.
Building an alert system is composed of two steps:

[1]http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html
[2]http://info.tuwien.ac.at/ingeo/research/imeas.htm


InGES Hillside Questionaire

Where: ____

Expert: ____

Date: ___________________________

Characteristics:

Vegetation: meadow pasturage / loose masses / solid rock / forest / single trees
Granual Material: block / rock / gravel / sand / silt / clay
Subsoil: malleable / hard
Pieces of Rock: yes recent / yes within 1 year / yes older than 1 year
Indicates: no / yes recent / yes within 1 year / yes older than 1 year
Slope Angle: low / medium / high / very high
Slope Profile Up: linear / convexly / concavely
Slope Profile Down: linear / convexly / concavely
Slip Surface: no / yes / 1m / 10m / 100m
Material Underground: gravel / clay / basalt / granit / shale
Fine Grit: high / very high
Saturation Soil: low / medium / high / very high / oversaturated
Leaning Trees: yes / no
Sickled Trees: yes / no
Leaning Rocks: yes / no
Crack: yes recent / yes overgrown / no
Rock Joint: not / medium / strong
Joint Orientation: dip / strike
Insolation: no / low / medium / high
Permafrost: yes / no / recent past
Movement Location: up / down / middle
Location of endangered Area: up / down / middle
Frost Thaw Cycle: yes / no
Depth Movement: shallow / deep
Endangered Area: streets / settlements / rivers
Risk Potential: low / lowmedium / medium / mediumhigh / high / very high

Figure 3.4: Categorization questionaire

1. Recognizing the relevant factors.

2. Detecting the correlation between them.

In highlighting the second step we have to look at the way how the risk is rated. This process is again divided into two steps:

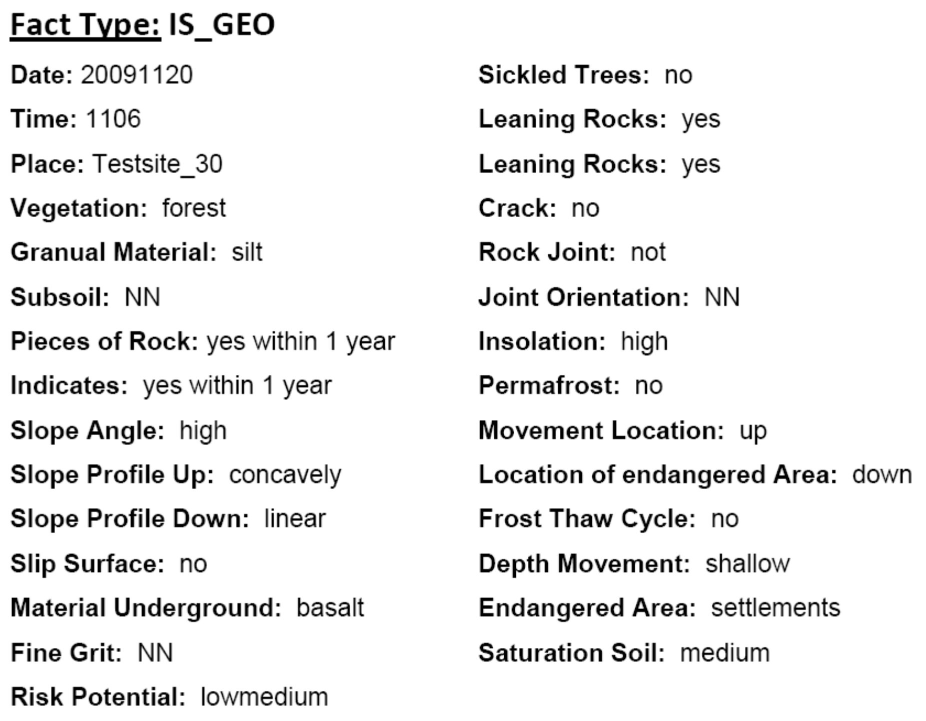
1. Identification of the *"Initial Risk Factor"*: This factor describes the assessment of the initial risk of a geological site (e.g., hillside).
2. Identification of the *"Dynamic Risk Factor"*: This factor describes the assessment of the temporal risk development.

3.2.1.1 Initial Risk Factor

This calculation process is performed once during the initial startup of InGES for a new geological site. The reason for this process is to assess the current risk factor of the site to be observed with InGES. In other words the risk factor prevalent during the assessment is captured. The basis for this calculation is a questionnaire (see Figure 3.4) about the hillside that is developed by the research group around InGES. It is given to experts in order to categorize the characteristics of the geological site. The purpose is to leverage expert knowledge to determine the kind and condition of the hillside.
However these expert assessments also generate a problem. The problem is that the questioned experts tend to variations[3]. For the same hillside, one expert might assess the slope angle as high whereas another expert states the slope angle is medium. InGES calculates the initial risk factor out of the selected characteristics of the questionnaire (see Figure 3.4). However since there exist variations the system has to deal with this circumstance. Especially disproportional influence (e.g., sharp risk increases/decreases) of minor characteristic changes are unwanted. In order to detect such sharp risk changes the decision was taken to add a behavior test tool to EM4J. This feature adds the possibility to perform automated tests in order to for example check the course of the initial risk assessment over all possible characteristic permutations. It gives the user the possibility to run InGES with various fictional geological sites (e.g., hillsides), calculate the initial risk factor and check the outcome. Later on in Chapter 4 we will see how these tests can be created and run in EM4J. Additionally we will see in Chapter 5 the magnitude of these variations and consequently the need for automated tests.
Another point we want to stress here is the question why decision trees are vital. Recall the exemplary rule trace from Chapter 1. The problem prevalent in this trace was the high amount of textual information. To ease the burden for the user a decision tree abstracts from this basis to a much more manageable graphical representation of the crucial information. Therefore we now take a closer look on the creation of the decision tree for the initial risk factor of some geological site.
This RBS has one fact in its factbase that represents the characteristics of this geological site (e.g., hillside). In the last slot of this fact the expert's risk estimation can be seen (`lowmedium-_1`).
The rulebase consists of 67 rules which calculate the initial risk factor. Highlighting all of these rules goes beyond the scope of this chapter therefore we only present the ones that are actually fired during this example:

```
MAIN::subsoil_malleable:
  IF data.granual_material = clay OR
```

[3]We already touched on this issue in Chapter 1.



Fact Type: IS_GEO

Date: 20091120	**Sickled Trees:** no
Time: 1106	**Leaning Rocks:** yes
Place: Testsite_30	**Leaning Rocks:** yes
Vegetation: forest	**Crack:** no
Granual Material: silt	**Rock Joint:** not
Subsoil: NN	**Joint Orientation:** NN
Pieces of Rock: yes within 1 year	**Insolation:** high
Indicates: yes within 1 year	**Permafrost:** no
Slope Angle: high	**Movement Location:** up
Slope Profile Up: concavely	**Location of endangered Area:** down
Slope Profile Down: linear	**Frost Thaw Cycle:** no
Slip Surface: no	**Depth Movement:** shallow
Material Underground: basalt	**Endangered Area:** settlements
Fine Grit: NN	**Saturation Soil:** medium
Risk Potential: lowmedium	

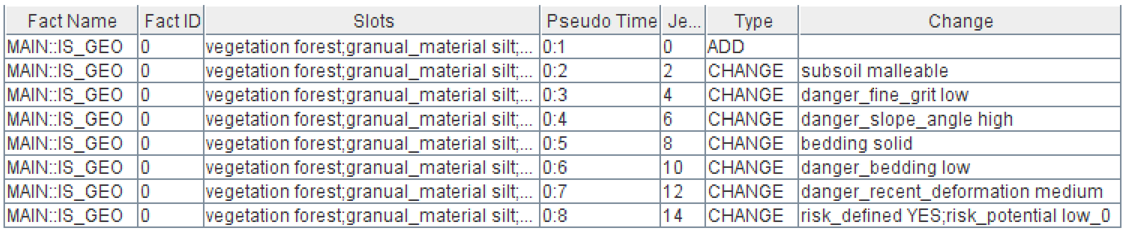
Figure 3.5: Fact for initial risk factor calculation

```
      data.granual_material = sand OR
      data.granual_material = silt
   THEN
      data.subsoil malleable



MAIN::recent_deformation_medium_c:
   IF (data.subsoil = malleable OR data.subsoil = hard) AND
      data.pieces_of_rock = yes_within1year AND
      data.indicates = yes_within1year
   THEN
      data.danger_recent_deformation medium



MAIN::slope_angle_high:
    IF (data.subsoil = malleable OR data.subsoil = hard) AND
       data.slope_angle = high
    THEN
```

```
      data.danger_slope_angle high



MAIN::bedding_no:
  IF (data.slip_surface = NN AND
     data.material_underground != NN) OR
     (data.slip_surface = NO AND
     data.material_underground != NN) OR
     data.material_underground = granit
  THEN
     data.bedding solid



MAIN::bedding_a:
  IF (data.slip_surface = 1m AND
     (data.material_underground = clay OR
     data.material_underground = basalt)) OR
     (data.slip_surface = 10m AND
     (data.material_underground = clay OR
     data.material_underground = basalt OR
     data.material_underground = shale)) OR
     (data.slip_surface = 100m AND
     (data.material_underground = clay OR
     data.material_underground = basalt OR
     data.material_underground = shale))
  THEN
     data.bedding prone_sliding



MAIN::bedding_low_a:
  IF data.bedding = solid
  THEN
     data.danger_bedding low



MAIN::danger_fine_grit_low_b:
  IF data.fine_grit = NN AND
     (data.saturation_soil = low OR data.saturation_soil = medium)
  THEN
```

```
      data.danger_fine_grit low



MAIN::IS_riskpot_rockfall_low_c:
  IF data.danger_recent_deformation = medium AND
      (data.danger_slope_angle = medium OR
      data.danger_slope_angle = high) AND
      data.danger_bedding = low AND
      data.danger_fine_grit = low AND
      (data.risk_potential != low_0 OR
      data.risk_potential != lowmedium_1 OR
      data.risk_potential != medium_2 OR
      data.risk_potential != mediumhigh_3 OR
      data.risk_potential != high_4 OR
      data.risk_potential != very_high_5) &&
      data.risk_defined != YES
  THEN
      data.risk_defined YES
      data.risk_potential low_0
```

Figure 3.6 and 3.7 show the fact history respectively the order of rule firing. The decision tree is based on this rule trace and fact history. The exact generation method of this RBS is now listed



Fact Name	Fact ID	Slots	Pseudo Time	Je...	Type	Change
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:1	0	ADD	
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:2	2	CHANGE	subsoil malleable
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:3	4	CHANGE	danger_fine_grit low
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:4	6	CHANGE	danger_slope_angle high
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:5	8	CHANGE	bedding solid
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:6	10	CHANGE	danger_bedding low
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:7	12	CHANGE	danger_recent_deformation medium
MAIN::IS_GEO	0	vegetation forest;granual_material silt;...	0:8	14	CHANGE	risk_defined YES;risk_potential low_0

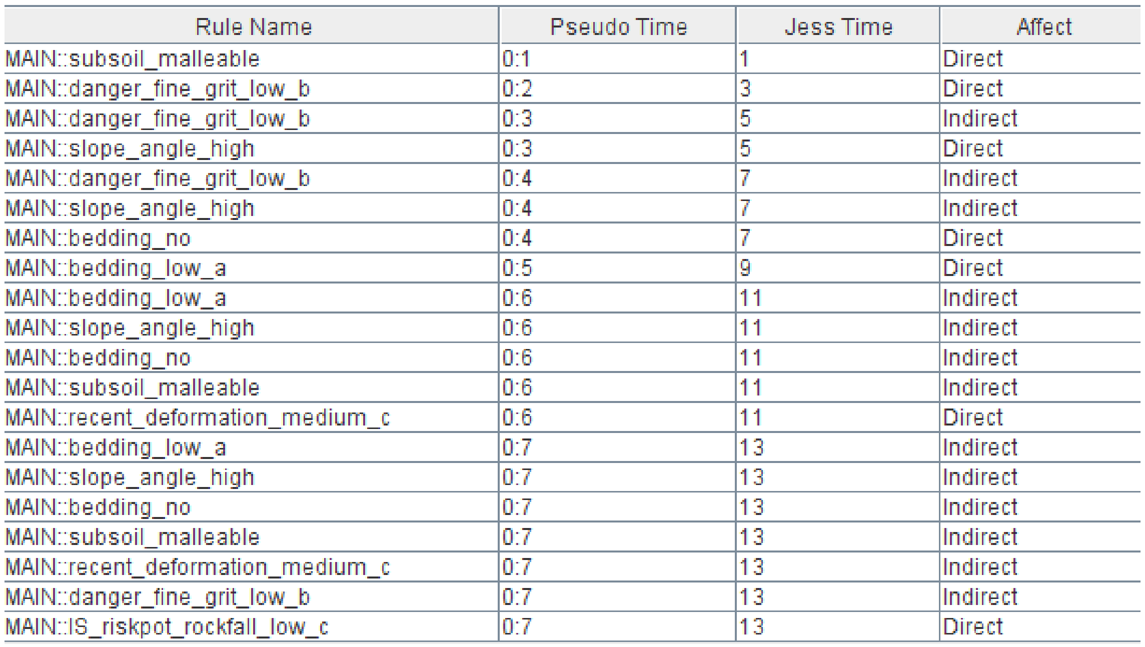
Figure 3.6: Fact history

below[4]:

Step 1 The initial fact leads to the firing of rule `MAIN::subsoil_mallable`. It changes the characteristic `subsoil` of the fact to `malleable`. This can be seen in the second line of Figure 3.6 in the column `Change`. The decision tree so far can be seen in Figure 3.8.

Step 2 The next rule fired is `MAIN::danger_fine_grit_low_b`. This one changes the characteristic `danger_fine_grit` to `low` which is symbolized through the node `MAIN ::IS_GEO:0:3 [CHANGE]`. Figure 3.6 shows this change in the third line with again

[4]We are not going into any geological details since this is not the topic of this thesis.



Rule Name	Pseudo Time	Jess Time	Affect
MAIN::subsoil_malleable	0:1	1	Direct
MAIN::danger_fine_grit_low_b	0:2	3	Direct
MAIN::danger_fine_grit_low_b	0:3	5	Indirect
MAIN::slope_angle_high	0:3	5	Direct
MAIN::danger_fine_grit_low_b	0:4	7	Indirect
MAIN::slope_angle_high	0:4	7	Indirect
MAIN::bedding_no	0:4	7	Direct
MAIN::bedding_low_a	0:5	9	Direct
MAIN::bedding_low_a	0:6	11	Indirect
MAIN::slope_angle_high	0:6	11	Indirect
MAIN::bedding_no	0:6	11	Indirect
MAIN::subsoil_malleable	0:6	11	Indirect
MAIN::recent_deformation_medium_c	0:6	11	Direct
MAIN::bedding_low_a	0:7	13	Indirect
MAIN::slope_angle_high	0:7	13	Indirect
MAIN::bedding_no	0:7	13	Indirect
MAIN::subsoil_malleable	0:7	13	Indirect
MAIN::recent_deformation_medium_c	0:7	13	Indirect
MAIN::danger_fine_grit_low_b	0:7	13	Indirect
MAIN::IS_riskpot_rockfall_low_c	0:7	13	Direct

Figure 3.7: Rule trace

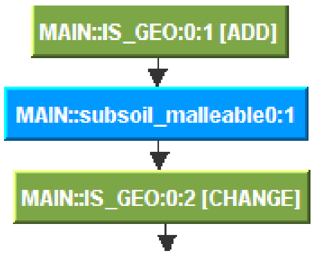


Figure 3.8: Initial risk factor calculation: Step 1

the changed slot displayed in the last column. The decision tree so far can be seen in Figure 3.9.

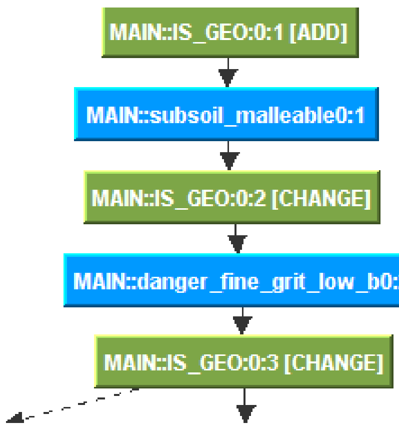

MAIN::IS_GEO:0:1 [ADD]
MAIN::subsoil_malleable0:1
MAIN::IS_GEO:0:2 [CHANGE]
MAIN::danger_fine_grit_low_b0:2
MAIN::IS_GEO:0:3 [CHANGE]

Figure 3.9: Initial risk factor calculation: Step 2

Step 3 In this step there are two rules fired. Rule `MAIN::slope_angle_high` and the previously fired `MAIN::danger_fine_grit_low_b`. Both rule's conditions are met, they are fired and their actions are performed. However just rule `MAIN::slope_angle_high` has an actual effect on the factbase because the characteristic `danger_slope_angle` is changed to `high`. On the other hand, the action from `MAIN::danger_fine_grit_low_b` (the change of the characteristic `danger_fine_grit` to `low`) is already in place and not changed since. Due to this circumstance the edge is dotted. Otherwise it would be solid. The decision tree so far can be seen in Figure 3.10.

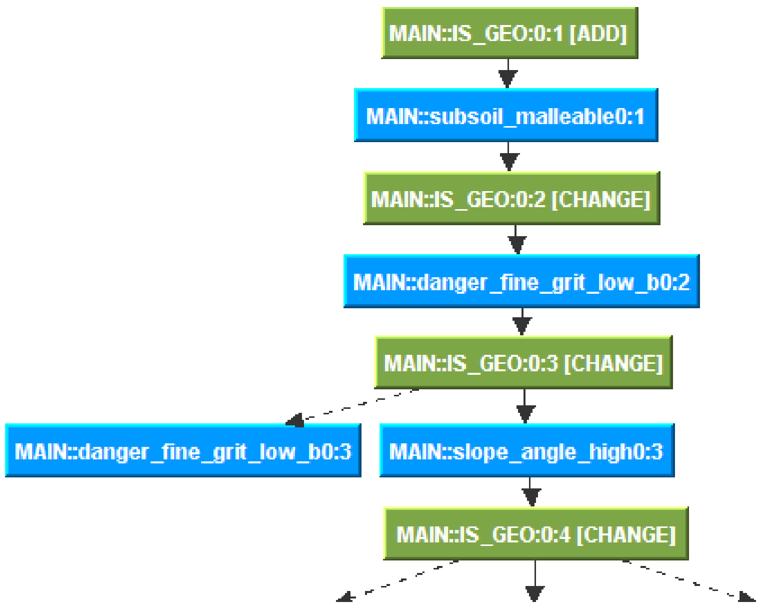

MAIN::IS_GEO:0:1 [ADD]
MAIN::subsoil_malleable0:1
MAIN::IS_GEO:0:2 [CHANGE]
MAIN::danger_fine_grit_low_b0:2
MAIN::IS_GEO:0:3 [CHANGE]
MAIN::danger_fine_grit_low_b0:3
MAIN::slope_angle_high0:3
MAIN::IS_GEO:0:4 [CHANGE]

Figure 3.10: Initial risk factor calculation: Step 3

Step 4 In this step the same situation occurs. The three rules (`MAIN::slope_angle_high`, `MAIN::danger_fine_grit_low_b` and `MAIN::bedding_no`) are fired. However only the last one actually changes the factbase. The action of rule `MAIN::bedding_no` sets the characteristic `bedding` to `solid`. The decision tree so far can be seen in Figure 3.11.

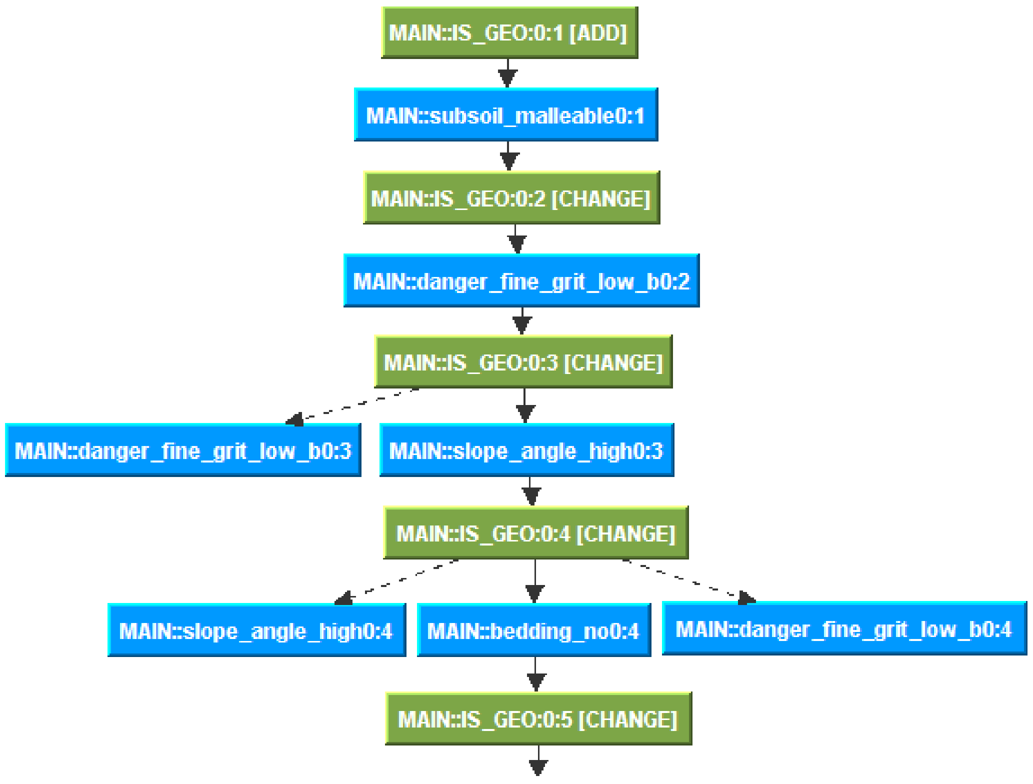

MAIN::IS_GEO:0:1 [ADD]
MAIN::subsoil_malleable0:1
MAIN::IS_GEO:0:2 [CHANGE]
MAIN::danger_fine_grit_low_b0:2
MAIN::IS_GEO:0:3 [CHANGE]
MAIN::danger_fine_grit_low_b0:3
MAIN::slope_angle_high0:3
MAIN::IS_GEO:0:4 [CHANGE]
MAIN::slope_angle_high0:4
MAIN::bedding_no0:4
MAIN::danger_fine_grit_low_b0:4
MAIN::IS_GEO:0:5 [CHANGE]

Figure 3.11: Initial risk factor calculation: Step 4

Step 5 Here only rule `MAIN::bedding_low_a` is fired. It changes the characteristic `danger_bedding` to `low`. More and more additional information such as connections between the geological characteristics of the hillside and the relevance of them in reference to the risk assumption are drawn. Every step takes the system closer to the final result. The decision tree so far can be seen in Figure 3.12.

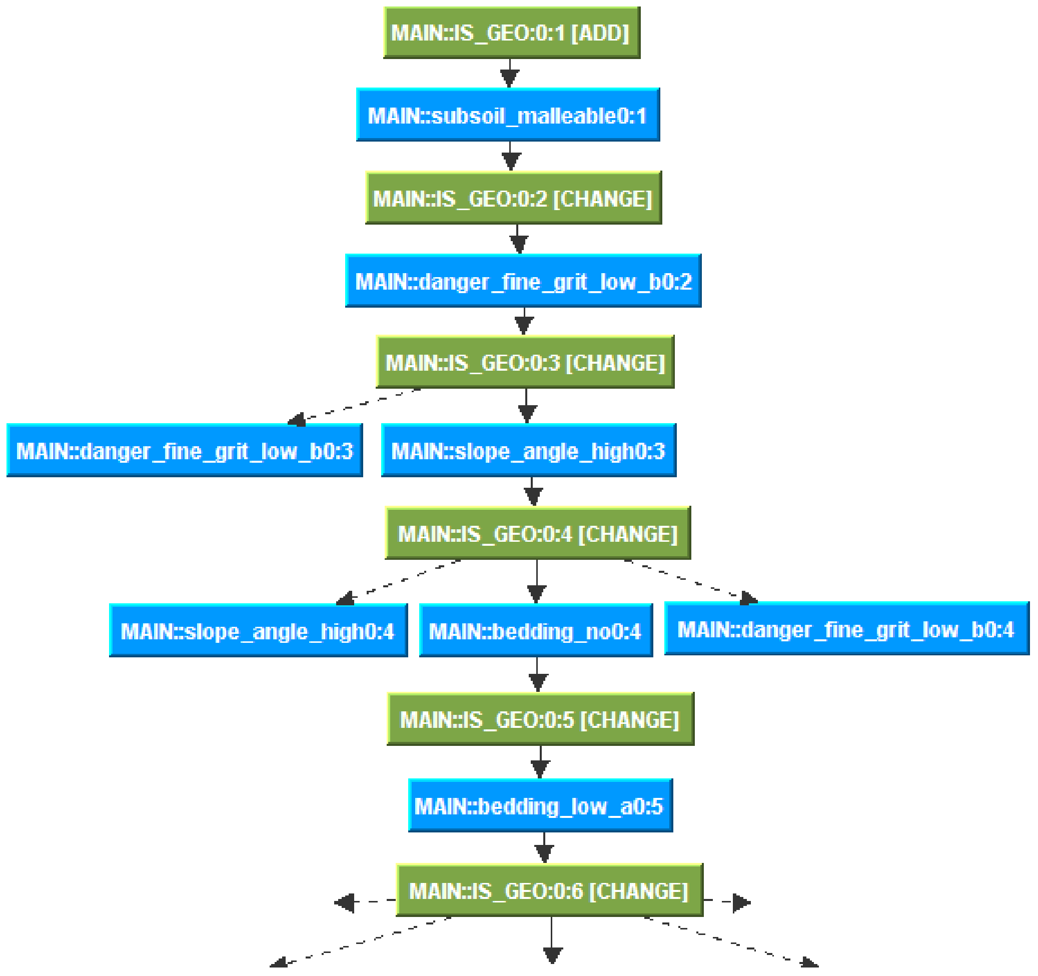

MAIN::IS_GEO:0:1 [ADD]
MAIN::subsoil_malleable0:1
MAIN::IS_GEO:0:2 [CHANGE]
MAIN::danger_fine_grit_low_b0:2
MAIN::IS_GEO:0:3 [CHANGE]
MAIN::danger_fine_grit_low_b0:3
MAIN::slope_angle_high0:3
MAIN::IS_GEO:0:4 [CHANGE]
MAIN::slope_angle_high0:4
MAIN::bedding_no0:4
MAIN::danger_fine_grit_low_b0:4
MAIN::IS_GEO:0:5 [CHANGE]
MAIN::bedding_low_a0:5
MAIN::IS_GEO:0:6 [CHANGE]

Figure 3.12: Initial risk factor calculation: Step 5

Step 6 Five rules are fired. Four of them were already fired in the past. The effects of their actions are already present in the factbase consequently the edges are dotted. On the other hand, the action of rule `MAIN::recent_deformation_medium_c` is not yet performed. It changes the characteristic `danger_recent_deformation` to `medium`. The decision tree so far can be seen in Figure 3.13.

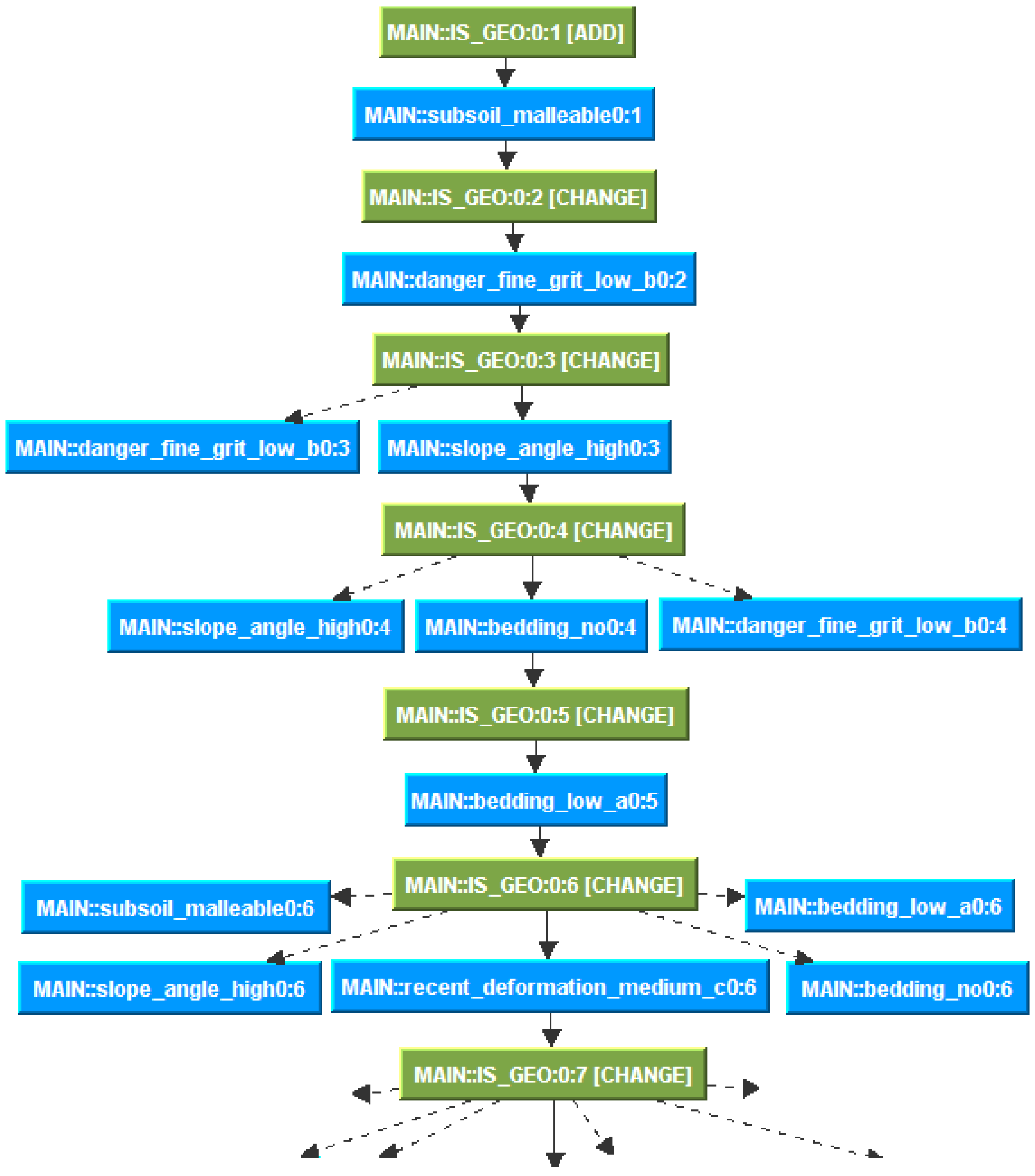

MAIN::IS_GEO:0:1 [ADD]
MAIN::subsoil_malleable0:1
MAIN::IS_GEO:0:2 [CHANGE]
MAIN::danger_fine_grit_low_b0:2
MAIN::IS_GEO:0:3 [CHANGE]
MAIN::danger_fine_grit_low_b0:3
MAIN::slope_angle_high0:3
MAIN::IS_GEO:0:4 [CHANGE]
MAIN::slope_angle_high0:4
MAIN::bedding_no0:4
MAIN::danger_fine_grit_low_b0:4
MAIN::IS_GEO:0:5 [CHANGE]
MAIN::bedding_low_a0:5
MAIN::subsoil_malleable0:6
MAIN::IS_GEO:0:6 [CHANGE]
MAIN::bedding_low_a0:6
MAIN::slope_angle_high0:6
MAIN::recent_deformation_medium_c0:6
MAIN::bedding_no0:6
MAIN::IS_GEO:0:7 [CHANGE]

Figure 3.13: Initial risk factor calculation: Step 6

Step 7 We are in the final step of the process. Among others the rule `MAIN::IS_riskpot_rockfall_low_c` is fired. This rule actually computes the final initial risk factor. Its actions terminate the process via setting the characteristic `risk_defined` to `YES`. The final decision tree can be seen in Figure 3.14.

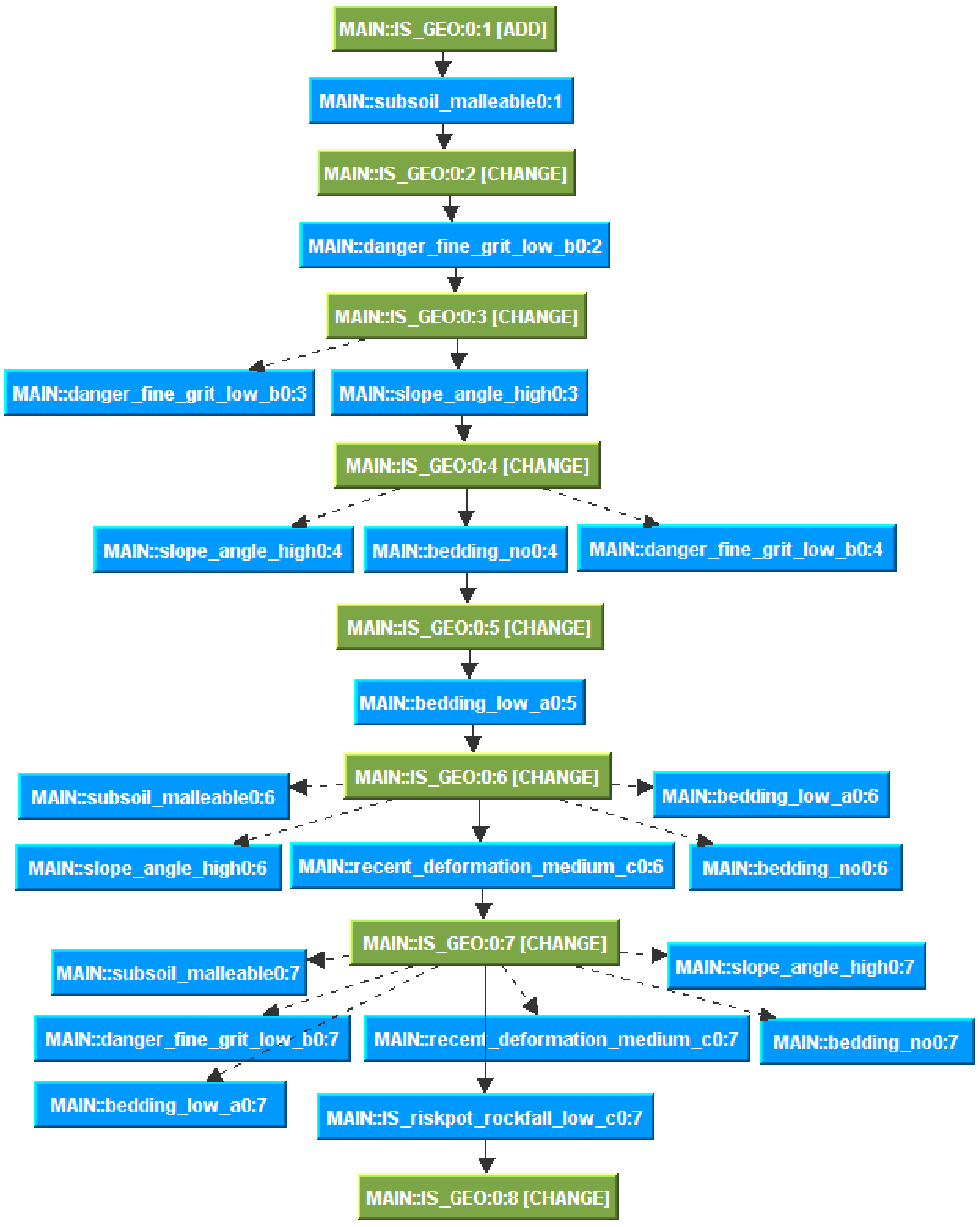

MAIN::IS_GEO:0:1 [ADD]
MAIN::subsoil_malleable0:1
MAIN::IS_GEO:0:2 [CHANGE]
MAIN::danger_fine_grit_low_b0:2
MAIN::IS_GEO:0:3 [CHANGE]
MAIN::danger_fine_grit_low_b0:3
MAIN::slope_angle_high0:3
MAIN::IS_GEO:0:4 [CHANGE]
MAIN::slope_angle_high0:4
MAIN::bedding_no0:4
MAIN::danger_fine_grit_low_b0:4
MAIN::IS_GEO:0:5 [CHANGE]
MAIN::bedding_low_a0:5
MAIN::subsoil_malleable0:6
MAIN::IS_GEO:0:6 [CHANGE]
MAIN::bedding_low_a0:6
MAIN::slope_angle_high0:6
MAIN::recent_deformation_medium_c0:6
MAIN::bedding_no0:6
MAIN::subsoil_malleable0:7
MAIN::IS_GEO:0:7 [CHANGE]
MAIN::slope_angle_high0:7
MAIN::danger_fine_grit_low_b0:7
MAIN::recent_deformation_medium_c0:7
MAIN::bedding_no0:7
MAIN::bedding_low_a0:7
MAIN::IS_riskpot_rockfall_low_c0:7
MAIN::IS_GEO:0:8 [CHANGE]

Figure 3.14: Initial risk factor calculation: Step 7

Compared to the rule trace, the essential steps and reasoning decisions are made visible in the decision tree. The trace lasts over eleven pages of text. No filtering is done and most important no visible differentiation in form of differing graphical characteristics (e.g., color, shape, etc.) between the rule leading to the final result and the other repeatedly fired is made. A lot of time is needed to investigate the system's decisions in order to come to the same result as presented by the decision tree. Here the user can almost instantly see why and how the initial risk factor is computed. Having only the rule trace, this understanding would take much longer to achieve.

3.2.1.2 Dynamic Risk Factor

This process is started periodically (e.g., once a hour). The outcome from the calculation of the initial risk factor serves as the basis for this *dynamic risk factor*. The initial risk factor is computed once during the startup of InGES for any new geological site. Therefore it can be considered a static assessment since it is only a snapshot in time. As the name suggests, there must be a dynamic component in this process. Actual data (e.g., from the last two weeks) such as information about the weather, deformation interpretation, etc. is progressively incorporated to compute the actual risk factor at the moment. The system delivers a continuous estimation of the likelihood of a geological change (e.g., landslide, rockfall, etc.).
During research and development of InGES knowledge engineers and domain experts are highly involved. In the application stage only domain experts will be the primary users.
The expertise of knowledge engineers is especially needed during the development phase. It is desired to incorporate enough domain knowledge into the system to compute risk factors of high quality. Later on in the system's application phase the domain expert uses his/her knowledge to validate the system's conclusion. This conclusion can either be accepted or rejected.

3.3 Rule Engine

Java Expert System Shell (Jess)[5] is a rule engine that is based upon the RETE algorithm [6] designed by Charles Forgy. However it uses an enhanced implementation[6]. Jess is fully implemented in Java[7] and build to process knowledge with the help of declarative rules. In short RETE is an effective algorithm to solve complex matching problems. The central idea of RETE is to store intermediate results computed during the process of matching facts to rules. This increases the time effectiveness of the rule engine at the expense of space utilization.
The integrated Java Event Framework from Jess is a strong advantage. It is very useful for tracking information about the reasoning performed. The type of events especially important to realize explanation capabilities are:

- **Fact Event:** This event has three categories:
  - **Add:** Thrown whenever a new fact is added to the factbase.
  - **Delete:** Thrown whenever an existing fact is removed from the factbase.

[5]For more detailed information we refer to http://www.jessrules.com
[6]http://www.jessrules.com/jess/docs/52/rete.html
[7]http://www.oracle.com/technetwork/java/index.html

- **Change:** Thrown whenever an existing fact is altered. In this case some attribute value is changed.

- **Rule Event:** This event has two categories:
  - **Activation:** Thrown whenever a rule has been activated or deactivated[8]. This option is not used by EM4J.
  - **Firing:** Thrown if a rule is fired. In detail the event is thrown before the rule's actions are performed.

With the help of these events we built the Rule Engine around Jess to monitor the core reasoning steps performed. We use internal time lines to give each event firing an unique timestamp. This architecture imposes an ordering on the collected data.
Along with collecting the concrete instances of facts at each time during the reasoning process and the fired rules we gather all rules, the templates as well as general information about the Jess program. At the end of each execution the data is saved into a new archive by passing it to the Knowledge Base Archive Manager via Java RMI.

3.4 Rule Dependency Analysis

This tool facilities the user in conveying the static relationships between rules contained in a rulebase. As a main feature, the user has the possibility to select certain rules and investigate their dependencies to one another. The selected rules are visualized by a graph. The graph displays the different categories of dependencies with different types of arcs that connect the nodes, which represent the rules. The dependencies are classified into four different categories:

1. **No Dependency:** In this case there is no connection from one rule to another. The graph displays the nodes without any connection in order to show to the user that no static dependency is present.

2. **Positive Dependency:** Whenever a rule is fired its performed actions possibly change the factbase. This altered factbase satisfies the conditions of at least one other rule unless a final state respectively recommendation is reached. This rule is then possibly fired. Here we now must distinguish between two subcategories:

   a) **Partial Dependency:** In this case the actions of rule `A` change the factbase with the result that some but not all of the conditions of some rule `B` are fulfilled. The graph displays this type of dependency as a dashed black arc between the source rule `A` and the sink rule `B`.

   b) **Full Dependency:** In this case the actions of rule `A` change the factbase with the result that all of the conditions of some rule `B` are fulfilled. This circumstance possibly leads to the rule's firing. The graph displays this type of dependency as a continuous black arc between the source rule `A` and the sink rule `B`.

[8]For details about the meaning of activation respectively deactivation we refer to http://www.jessrules.com/jess/docs/71/rete.html#

3. **Negative Dependency:** In this special case we want to highlight the circumstance that the actions of some rule `A` lead to a state which absolutely dissatisfies some or all of the conditions of some rule `B`. In other words if rule `A` is fired rule `B` will never be fired immediately afterwards. The graph displays this type of dependency as a continuous red arc between the source rule `A` and the sink rule `B`.

3.4.1 The Process of Dependency Analysis

We are going to look at the process building the dependency graph. The algorithm within EM4J uses a 3-valued logic (`TRUE`, `FALSE`, `UNKNOWN`) to determine whether the rule conditions in connection with the actions of other rules would be true or false during execution. The third value (`UNKNOWN`) is used for complex assignments like variables, functions, subprograms, etc. These complex assignments are not evaluated. They are skipped by the algorithm and the value `UNKNOWN` is assigned.
In summary this algorithm to some extend mirrors the matching performed by the rule engine. However it is not the full spectrum of rule matching considering functions, variables, user input, etc. is performed. Instead a static dependency analysis is realized. The following example will explain this analysis.
Consider the following rules:

```
Rule 1: accerlerate
  IF
    (human.right_foot = on_pedal)
  THEN
    (car.status = accelerate)

Rule 2: open_throttle
  IF
    (car.status = accelerate)
  THEN
    (car.throttle = open)
```

Rule `open_throttle` can be fired when the status of the car is set to `accelerate`. The rule `accelerate` is doing exactly this. Consequently these two rules possess a full static dependency.
One concern of such an algorithm is not to reach the time and resource effort of a full matching (matching considering functions, variables, user input, etc.) like the RETE algorithm is doing during the execution. The complexity of this algorithm has to be reduced. This is achieved through the restriction that only static assignments are checked. There is no fact evaluation done on variables, functions, etc., since they are only assigned/calculated during the execution. Therefore the 3-valued logic is in place. Whenever the algorithm discovers a variable, function, etc., the value `UNKNOWN` is assigned. The algorithm solely works with the program code of the rules. In other words the conditions are statically matched to the actions.

```
 1:for all selected rules
 2:  Take the next rule r
 3:  Filter all modifying actions from the RHS of r
 4:  for all other rules
 5:    Take the next rule n
 6:    for all modifying actions
 7:      Take the next modifying action a
 8:      for all conditions on the LHS of n
 9:        Take the next condition c
10:        Check the truth value of c in connection with a
11:      end for
12:    end for
13:  end for
14:  Filter positive and negative dependent rules
16:end for
```

This pseudo code illustration of the algorithm presents the general technique for identifying the existing dependencies. We are going to look at two specific functions in the algorithm in detail. The function ’`Check the truth value of c in connection with a`’ and ’`Filter positive and negative dependent rules`’. The former examines the condition forest. Internally the LHS of every rule is parsed into a forest (at least one tree) according to its structure. Figure 3.15 illustrates this procedure by showing an example condition and its corresponding tree. The check function actually starts with the root node of every tree in the forest (see Line `8`) and recursively traverses the tree in order to determine the truth value.

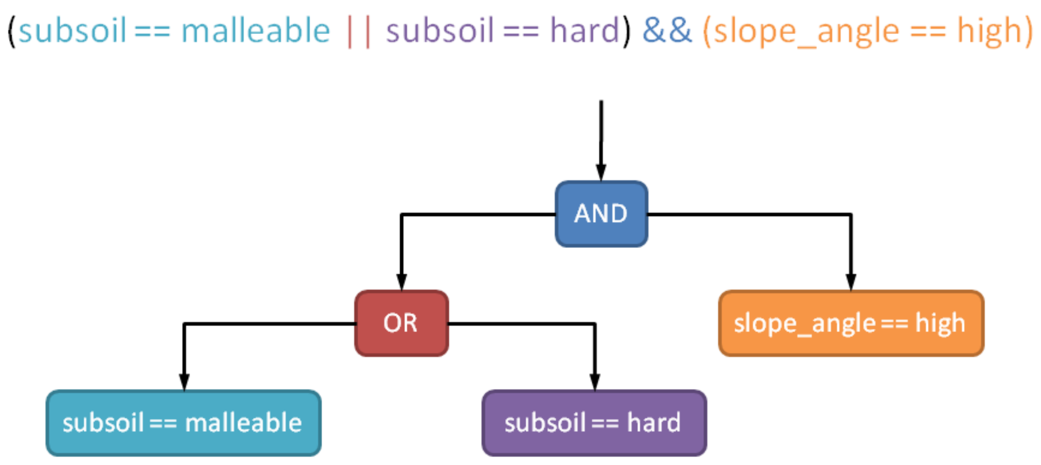

(subsoil == malleable || subsoil == hard) && (slope_angle == high)
AND
OR
slope_angle == high
subsoil == malleable
subsoil == hard

Figure 3.15: Tree parsing of LHS: A condition (above) and its parsed tree (below)

After all trees in the forest are checked the second function to be highlighted is applied. This function has the functionality of a filter and is shown at the end of the algorithm. This filter distinguishes between the following conditions:

1. If the result is `TRUE` then there is a full dependency present.
2. If the result is `UNKNOWN` and at least one condition is met then there is a partial dependency present.
3. If the result is `UNKNOWN` and no condition is met then there is no dependency present.
4. If the result is `FALSE` then there is a negative dependency present.

The data generated by the dependency analysis is again handled by a forest structure. As a last step this dependency forest is then used for visualization.

3.5 Decision Tree Analysis

The core investigation tool is the Decision Tree Analyser with the capability to explore the rule trace in connection with the facts present in the KBS. A decision tree is created starting with the initial facts, going through the reasoning process and ending with the final recommendation. The raw data collected by the Rule Engine during the execution of the KBS constitutes the input for this process. In a second step based on this input an internal intermediate representation of the decision tree is computed. This tree is used as the input for the visualization process. The last step is visualizing the tree to the user. With the help of layout algorithms[9] that are incorporated into EM4J the decision tree is graphically presented to the user. The components of the graphical visualization are described in Table 3.1.
For better understanding how the different visual components are used consider the example from Section 1.1. This example highlights the different arcs and nodes that are available in the decision tree. For the sake of completeness we again list the rulebase that contains three rules and the factbase that contains one fact:

```
    (MAIN::data (slot_a 1) (slot_b nil) (slot_c nil))

MAIN::match-1_:  IF (data.slot_a=1) THEN (data.slot_b 2)
MAIN::match-2_:  IF (data.slot_b=2) THEN (data.slot_c 3)
MAIN::match-3_:  IF (data.slot_c=3) THEN ( print "END" )
```

Let us have a look at the last step of this example. It represents the final decision tree (see Figure 3.16). The last change to the factbase is represented through the node `MAIN::data:`
`0[CHANGE]`. The name of this node is composed of the Jess module the fact is placed, the name of the template, the unique identifier of the fact and the type of factbase modification (in this case a change to a certain slot). This factbase change has led to the firing of `MAIN::match-1_`, `MAIN::match-2_` and `MAIN::match-3_`. These firings are all displayed as redundant firings (edges are displayed as dashed) since their actions do not change the factbase. For example consider rule `MAIN::match-1_`. Its action is to change the slot `slot_b` of the fact used to

[9]For more information we refer to Section 4.4.2.3.

Types	Details
	Nodes
Rules	Labeled with their name Tooltips display the conditions (LHS) and actions (RHS)
Facts	Labeled with their name Tooltips display (slot value) pairs and potentially the change triggered through the actions of the last fired rule
	Continous Arcs
Fact $\longrightarrow$ Rule	Impact Firing: The rule is triggered through the attributes of the fact that fulfill the rule's conditions. The actions of the rule have an affect on the factbase
Rule $\longrightarrow$ Fact	Impact Firing: The fired rule's actions change the facts attributes
	Dashed Arcs
Fact $\dashrightarrow$ Rule	Redundant Firing: The rule is repeatedly triggered by the attributes of the fact that fulfill the rule's conditions. This case arises whenever the rule's conditions are still met later on in the reasoning process. However the actions of the rule do not have any effect to the factbase because the changes provoked by them are already in place. In other words the redundant firing of rule does not change the global state of the system.

Table 3.1: Decision tree components

fulfill the condition to `2`. But this slot is already set to this value. Therefore no change happened to the factbase. This is also true for rule `MAIN::match-3_` which is fired for the first time. Its action does not change the factbase. It just writes text to the output.
Additionally to this graphical decision tree the fired rules as well as the history of fact changes are presented to the user in form of separate tables.

3.5.1 Decision Tree Generation

The algorithm generating the decision tree uses the fired rules and the history of fact changes as an input. The output is an internal tree representation of the reasoning process which is then used for creating the graphical representation seen by the user.

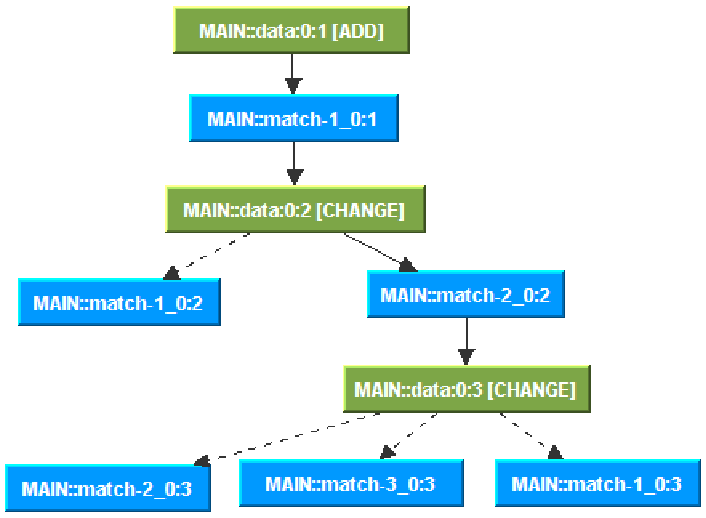


Figure 3.16: Last step from the example out of Section 1.1

Since we track the data with the Rule Engine we have the possibility to add timestamps to each Java Event triggered by the firing of a rule or a change to the factbase. In fact this step already represents the first action during the generation of the decision tree. The architecture of this time reckoning is the following. Every added fact gets its own timeline. The actual timestamp is composed of the fact identifier followed by a colon and an ascending number.
Rules on the other hand are handled differently in this timeline architecture. Rules do not have their own timeline. However every fired rule is caused by at least one fact. The timestamp of a rule is composed as a list of the unique timestamp(s) of the fact(s) that cause the rule's conditions to be fulfilled. Therefore this structure defines the dependencies between rules and facts through the fact timelines. This genuine structure simplifies the generation of the decision tree. Negative conditions are not explicitly considered by EM4J. Therefore negative and positive conditions are graphically not distinguished.
The principle working method of this algorithm is shown in the following pseudo code[10]. In this algorithm we talk about fired rules `r`. These rules are actual rule instances out of the Rule Engine. They are composed of the rule itself as well as the facts that lead to the firing.

```
getTree() {
 for all fired rules r
   if r is not processed
     build(r)
   end if
 end for
}
```

[10]Note: Facts that fulfill at least one condition of rule r are called fulfilling facts.

```
01:build(rule r) {
02: get all fulfilling facts from r
03: for all fulfilling facts f
04:   if f has value changes
05:     for all value changes c of f
06:       Get all affected rules
07:       for all affected rules a
08:         if a is not processed
09:           build(a)
10:         end if
11:       end for
12:     end for
13:   else
14:     Get asserting rule for f
15:     Take asserting rule a
16:     if a is not processed
17:       build(a)
18:     end if
19:   end if
20: end for
21:}
```

We highlight three important methods used within this algorithm.

1. `Get all fulfilling facts:`
   This function returns all the facts that possess the same timestamp as the rule itself. We already stated that the timestamp of each rule is composed of a list of fact timestamps that lead to the rule's condition(s) to be fulfilled. Therefore timestamp levels are introduced that host at least one fact as well as one rule. The facts are on this level because of their timeline definition. The rules on the other hand are on this level because of their dependency to the fact. This dependency mirrors the circumstance that at least one value of the fact satisfies at least one condition of the rule.
   Since we actually start this algorithm from the last rule backwards to the first we need to take all the facts on the same level because they are the predecessors in the reasoning process.

2. `f has value changes:`
   This check examines if the fact `f` has at least one slot value changed from the last reasoning step to this one. A reasoning step denotes the increment of one fact's unique timeline. In other words the actions of a rule have changed the information stored in this fact.

3. `Get all affected rules:`
   This function returns the rules that caused the change from the fulfilling fact determined earlier (see Line 4). We have to take the entries of one level above of the fact's timeline

(chronologically earlier). This leads the algorithm to the next rule to start going backward until the beginning of the reasoning process.

4. `Get asserting rule for f:`
   The function possibly returns a rule that asserted the fulfilling fact[11] or null if the fact is user created. Rules possess the ability to not only change facts in the factbase but also to create new ones. This special case has to be taken care of because within our timestamp architecture there would be no connection from the rule fired to the newly created fact. The reason for this is that every newly created fact gets its own timeline.

Additional notes:

- The processing of a rule is finished only if all its timestamps (the ones of the facts fulfilling the rule's conditions) are processed. Only then all its branches in the decision tree are generated.
- The term `value changes` in Line 5 of method `build()` refers to the modifications done to the fact by the last rule fired.

[11]A fulfilling fact is a fact that fulfills at least one condition of a rule.

CHAPTER 4

EM4J - The User Point of View

4.1 Overview

In the last chapter EM4J's architecture was highlighted. We saw the particular tools available, how they work and how they operate as a unit. This chapter describes the handling of EM4J and its features in detail. First and foremost this chapter presents the following two aspects, namely

1. highlighting the functions that the system offers and
2. describing how these functions are used.

EM4J is a toolbox that consists of four distinct but still coherent subsystems:

- Knowledge Base Archive Manager/Test Case Archive Manager
- Rule Engine
- Decision Tree Analyser
- Rule Dependency Analyser

The next sections delineate them in detail.

4.2 Knowledge Base and Test Case Archive Manager

These two archive managers are implemented as background jobs managing the data of the whole toolbox. The managers do not possess any graphical user interface.
The Knowledge Base Archive Manager's job is to administrate the data gathered by the Rule Engine with each and every execution. Via RMI the data is distributed to the explanation module for a further analysis of the data.
The Test Case Archive Manager has the same functionality as the Knowledge Base Archive

Manager. But instead of managing the information of the KBS itself, these archives contain the data collected by running the original program with a different setting. Later in this chapter we will describe this setting and its purpose in more detail.

4.2.1 Archive Types

EM4J possesses two types of archives, the knowledge base archive and the test case archive. The first contains real reasoning information (e.g., rule traces, fact traces, etc.) collected during the execution of a KBS. The second contains reasoning information too. However it is collected during the execution of a test case configuration. In other words the input this KBS executes does not come from real data it has the sole purpose to test some specific condition.

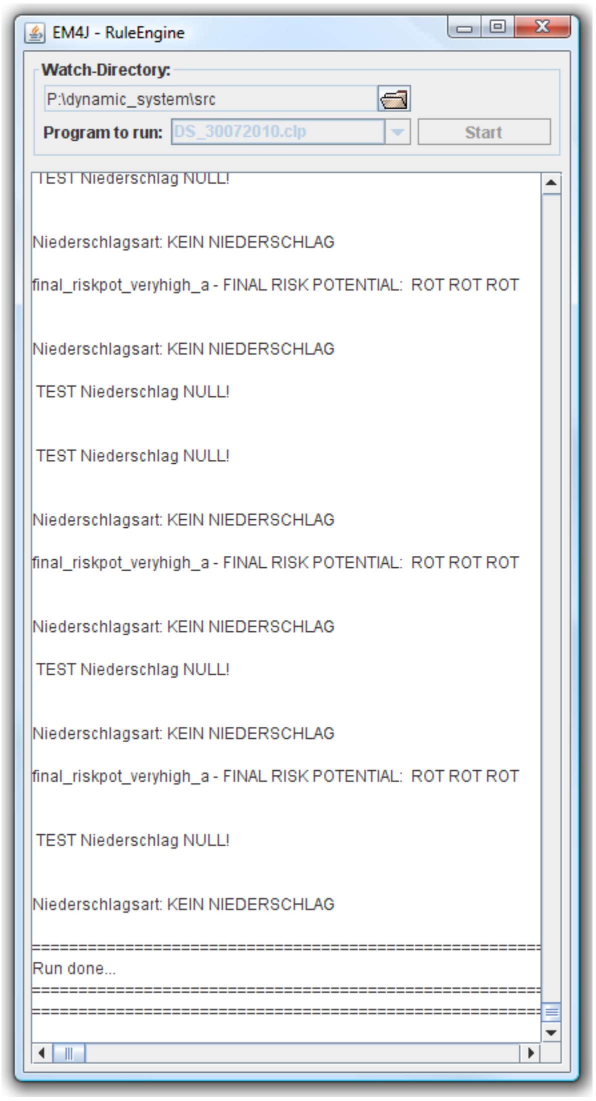
4.3 Rule Engine

The Rule Engine is the reasoning tool within the toolbox. It executes KBSs implemented in Jess and gathers information about these executions for later analysis. Each RBS implemented in Jess that is executed with the Rule Engine must not contain run or save-facts commands. Additionally every load-facts command must contain the fully qualified path of the file containing the facts. Otherwise the execution would result in an error.
There are two versions of the Rule Engine implemented. The first version is simple console based without a graphical user interface (GUI). The main purpose of it is to create the initial risk factor of a geological site observed with InGES. However the Rule Engine is in principle not restricted to running InGES. Many other KBSs implemented in Jess can be executed as well as explained. During startup two parameters have to be given. The fully qualified path of the directory containing the KBS and the name of the KBS itself.
The second version possesses a GUI. Its main purpose is to compute the dynamic risk factor. In order to fulfill this goal it is implemented as a background job. Starting and closing is done via a system tray item. We explain this version in the following section in detail.

4.3.1 Graphical Rule Engine

The main GUI of this tool can be seen in Figure 4.1. At the top there is a text field with a file chooser next to it. With this file chooser a directory where the Rule Engine listens to file changes can be selected. The main purpose of this is to recognize changes to input files. If the file is changed (e.g., saved with new content) the KBS is triggered to compute the dynamic risk factor. Below the upper graphical input mask a combo box is displayed. It enables the user to select a KBS to be executed. The combo box list is composed of KBSs that are contained in the before chosen watch directory. By clicking the button *Start* [1] the currently selected combo box entry is started. When the execution is finished the program is in a waiting mode until another file change happens.

[1]Buttons are formatted as follows: *Button Name* .

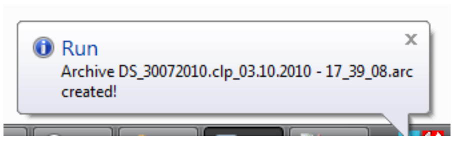


```
P:\dynamic_system\src
DS_30072010.clp

TEST Niederschlag NULL!

Niederschlagsart: KEIN NIEDERSCHLAG

final_riskpot_veryhigh_a - FINAL RISK POTENTIAL:  ROT ROT ROT

Niederschlagsart: KEIN NIEDERSCHLAG

 TEST Niederschlag NULL!

 TEST Niederschlag NULL!

Niederschlagsart: KEIN NIEDERSCHLAG

final_riskpot_veryhigh_a - FINAL RISK POTENTIAL:  ROT ROT ROT

Niederschlagsart: KEIN NIEDERSCHLAG

 TEST Niederschlag NULL!

Niederschlagsart: KEIN NIEDERSCHLAG

final_riskpot_veryhigh_a - FINAL RISK POTENTIAL:  ROT ROT ROT

 TEST Niederschlag NULL!

Niederschlagsart: KEIN NIEDERSCHLAG

=====================================================
Run done...
=====================================================
=====================================================
```

Figure 4.1: Graphical rule engine main view

Below the watch directory configuration panel displayed at the top of the GUI a text area is shown. This text area lists the output of each execution of the Jess rule engine. The syntax of the output is the following (The text in quotation marks represents actual output from the Rule Engine. All the other text is representative for the output written by the KBS. An example is displayed in Figure 4.1):

```
          The debug output of the Jess rule engine
==============================================
The textual output of the rules contained in the Jess program
==============================================
```



Run
Archive DS_30072010.clp_03.10.2010 - 17_39_08.arc created!

Figure 4.2: Popup message shown after the run

“Run done...”
==
==

Each time the KBS is executed the Knowledge Base Archive Manager creates an archive. This archive contains the reasoning information of the Rule Engine as well as additional data. The various other tools access these archives in order to explain the KBS results.
The name of this archive is given as a feedback message in the system tray (see Figure 4.2). Usually a program is shut down when the user closes the GUI window via the control buttons in the upper right corner. The Rule Engine on the other hand is not shut down. It still runs in the background. This is visualized by the icon in the system tray. Figure 4.2 exemplifies this circumstance. The Rule Engine can only be closed by clicking the right mouse button over the icon in the system tray. A popup menu is shown listing an *Exit* [2] entry, which shuts down the Rule Engine (see Figure 4.3).

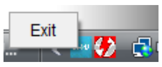


Figure 4.3: Exit menu of the rule engine

4.4 EM4J - The Toolbox

Straight after startup of EM4J the main GUI window of the program (depicted in Figure 4.4) is displayed on the desktop.

4.4.1 EM4J - Startup Window

This is the entry point into the system where the user can navigate through the various archives stored in the file system. The startup window consists of two parts. The left hand side shows the archives currently loaded (knowledge base or test case archives). Whereas the right hand side shows general information about a specific archive (see Figure 4.4). The general information is composed of the following list.

- The archive name,

[2]Menus are formatted as follows: *Menu Name* .

- the archive date,
- the number of templates,
- the number of initial facts,
- the number of rules in the original KBS,
- the number of fired rules, and
- the number of fact changes during the run.

In order to load a specific archive one of the archives in the tree on the left has to be selected with the mouse cursor. Since the loading time depends on the archive size a progress bar is shown. When this process is finished the button *Decision Tree* is enabled. If this button in the right bottom corner is clicked the system starts to compute the decision tree.
Now we want to highlight the other options present in the startup window before going a step further into the system by pushing the button *Decision Tree* .

4.4.1.1 Deleting Archives

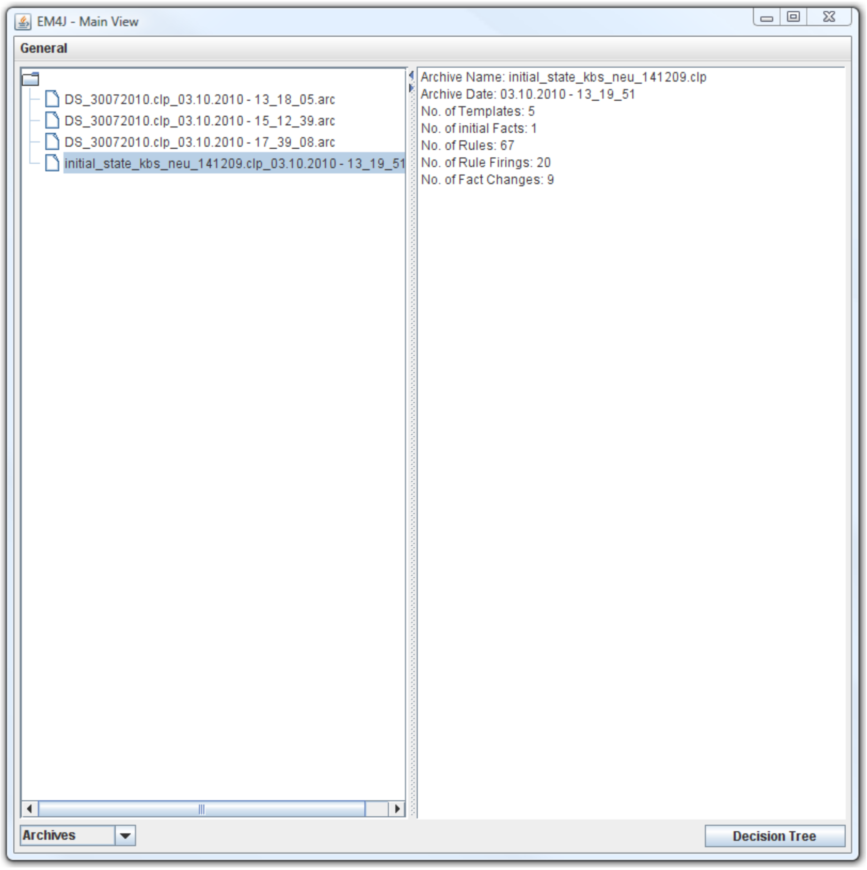
Deleting archives can be performed in two ways:

1. **Delete All Archives:** Physically delete the archive folder in the EM4J working directory in the file system in connection with a restart of the Knowledge Base Archive Manager. In the current version of EM4J there is no such feature implemented.
2. **Selectively Delete Archives:** Delete a selected archive in the startup window via the context menu. This context menu is opened by clicking the right mouse button with the mouse over the archive to delete (see Figure 4.5).

4.4.1.2 Test Case Archives Sources

Through the menu item *General/Options* it is possible to manage the test case archive sources. These sources are the file system directories containing the test case archives. The directory structure of these archives can be seen in Figure 4.6. Due to EM4J's internal processing the user has to select the directory named "testCaseArchives". This is the superior directory containing all test case archive directories. The reason for this structure is that the user need not select each archive directory on its own. Instead all archive directories are automatically loaded. With this function every stored archive (e.g., backups, test cases from other users, etc.) can be loaded into the system. Figure 4.7 shows the GUI displaying this option.
Adding and removing of the sources is performed by context menu interaction. There are two versions of the context menu:

1. There already exists at least one source in the system and the context menu is opened by clicking with the right mouse button on one of the table entries (see Figure 4.8). In this case the context menu shows two items, namely *Add* and *Remove* .



DS_30072010.clp_03.10.2010 - 13_18_05.arc
DS_30072010.clp_03.10.2010 - 15_12_39.arc
DS_30072010.clp_03.10.2010 - 17_39_08.arc
initial_state_kbs_neu_141209.clp_03.10.2010 - 13_19_51

Archive Name: initial_state_kbs_neu_141209.clp
Archive Date: 03.10.2010 - 13_19_51
No. of Templates: 5
No. of initial Facts: 1
No. of Rules: 67
No. of Rule Firings: 20
No. of Fact Changes: 9

Figure 4.4: EM4J - Main view with a loaded archive

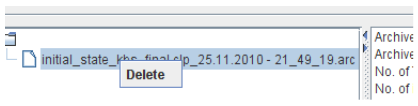


Figure 4.5: EM4J - Main view with context menu to delete an archive

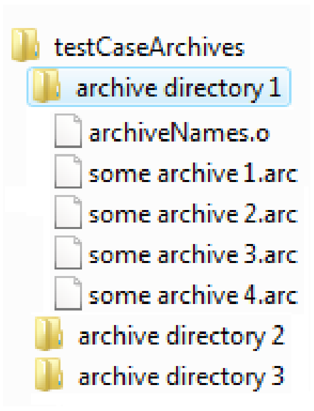


Figure 4.6: Test case archive structure

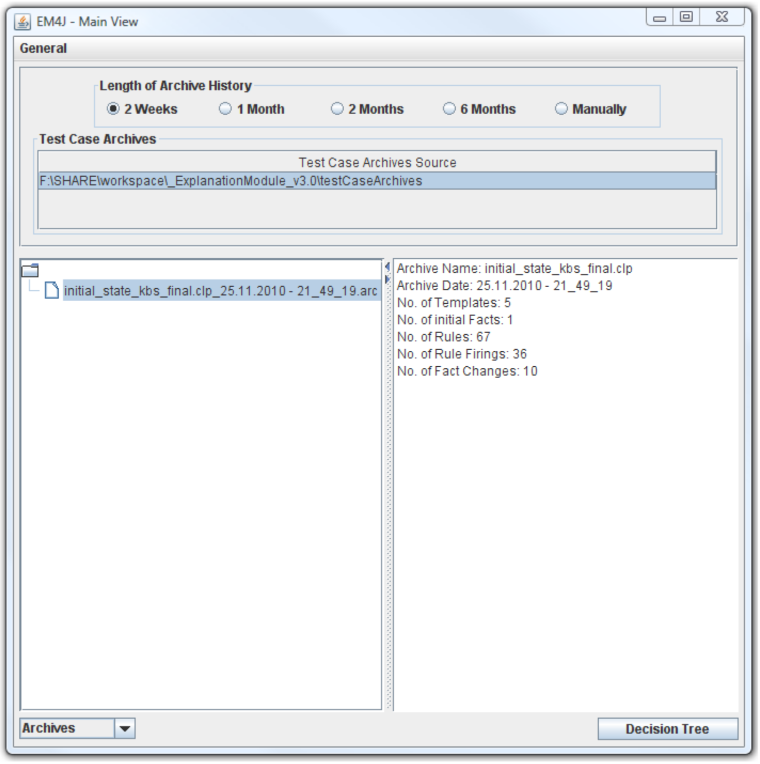
2. There is no source listed or although there exists at least one source the context menu is opened by clicking outside the table entries (see Figure 4.9). In this case the context menu has only one item, *Add* .

Adding a new test case archive source to the system is done by clicking *Add* in the context menu. Subsequently a file chooser is opened (see Figure 4.10) to select a "testCaseArchives" directory. Removing a test case archive source is done by selecting the target source and subsequently clicking *Remove* in the context menu.

4.4.1.3 Archive History Length

The archive history length specifies the length of the time period in which the created KBS execution archives are stored. This setting is used during the creation of a new archive by the Rule Engine. When a new archive is created the system checks whether older ones should be deleted.
This setting can be changed through the options shown by clicking the *General/Options* menu item (see Figure 4.7). The meaning of these five options are described below:

- **2 Weeks:** The archives created in the last two weeks are going to be stored.
- **n Months (n = 1,2,6):** The archives created in the last n months are going to be stored.
- **Manually:** The archives created are always going to be stored. The user has to delete them manually. Two options that fulfill this task are available.
  1. **Delete All Archives:** Physically delete the archive folder in the EM4J working directory on the file system in connection with a restart of the Knowledge Base Archive Manager. In the current version of EM4J this feature is not implemented.
  2. **Delete Archives Manually:** Manually delete archives in the startup window (see Section 4.4.1.1).


EM4J - Main View
General
Length of Archive History
2 Weeks
1 Month
2 Months
6 Months
Manually
Test Case Archives
Test Case Archives Source
initial_state_kbs_final.clp_25.11.2010 - 21_49_19.arc
Archives
Decision Tree

F:\SHARE\workspace_ExplanationModule_v3.0\testCaseArchives

Archive Name: initial_state_kbs_final.clp
Archive Date: 25.11.2010 - 21_49_19
No. of Templates: 5
No. of initial Facts: 1
No. of Rules: 67
No. of Rule Firings: 36
No. of Fact Changes: 10

Figure 4.7: EM4J - Main view with option pane folded out

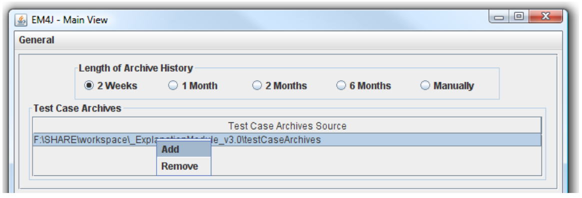

EM4J - Main View
General
Length of Archive History
2 Weeks
1 Month
2 Months
6 Months
Manually
Test Case Archives
Test Case Archives Source
F:\SHARE\workspace_Expl...e_v3.0\testCaseArchives
Add
Remove

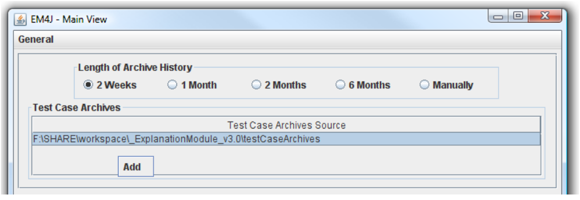
Figure 4.8: EM4J - Main view with option pane folded out and context menu

4.4.1.4 Switching the Archive Type

The combo box in the left bottom corner has the function to choose between "Archives" and "Test Cases". These are the synonyms for the two archive types in the system. The former symbolizes the knowledge base archives. The latter symbolizes the test case archives.

4.4.2 Decision Tree Analyser

Pushing the *Decision Tree* button starts the process of generating the decision tree as well as displaying its computed graphical visualization (see Figure 4.11). This window is divided into three parts:


EM4J - Main View

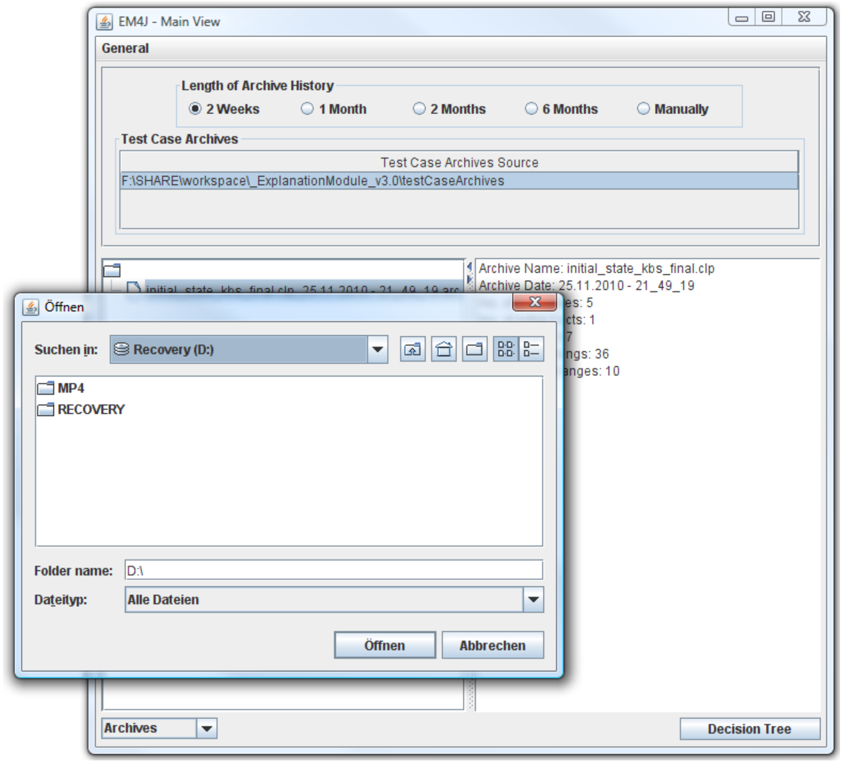
General

Length of Archive History: [✓ 2 Weeks] 1 Month 2 Months 6 Months Manually

Test Case Archives

Test Case Archives Source
F:\SHARE\workspace_ExplanationModule_v3.0\testCaseArchives

Figure 4.9: EM4J - Main view with option pane folded out and context menu


EM4J - Main View

General

Length of Archive History: [✓ 2 Weeks] 1 Month 2 Months 6 Months Manually

Test Case Archives

Test Case Archives Source
F:\SHARE\workspace_ExplanationModule_v3.0\testCaseArchives

initial_state_kbs_final.clp_25.11.2010 - 21_49_19.arc

Archive Name: initial_state_kbs_final.clp
Archive Date: 25.11.2010 - 21_49_19
es: 5
cts: 1
7
ngs: 36
anges: 10

Öffnen

Suchen in: Recovery (D:)

MP4
RECOVERY

Folder name: D:\
Dateityp: Alle Dateien

Figure 4.10: EM4J - Main view with test case archives source file chooser opened

1. **Fact History:** This list displays every factbase modification recorded during the reasoning procedure of the KBS. Every entry in the table represents one manifestation of a fact at a particular time. In other words each entry of the Fact History is a unique occurrence. An entry consists out of seven columns with the following meaning:

    a) **Fact Name:** The name of the underlying template. Templates define the structure of facts. It virtually represents their construction plan containing the slot names, the value type, the allowed values per slot, etc.

    b) **Fact ID:** The system-wide internal unique identifier of the fact.

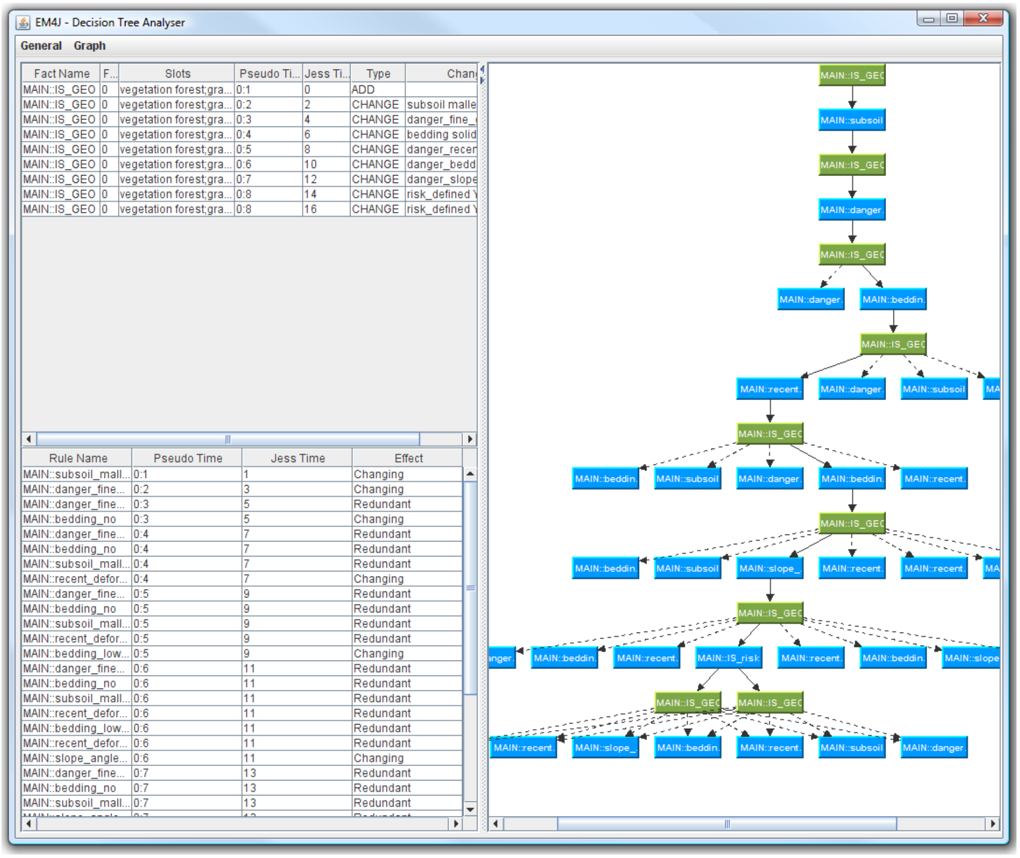
    c) **Slots:** This column contains the (slot value) pairs of the fact. The pairs are separated by semicolons, the slot names and values are separated by whitespaces.

  d) **Pseudo Time:** This is an internal EM4J timestamp given to the fact at the moment of its appearance. These timestamps build the basis for generating the decision tree. A more detailed presentation of this timeline architecture has been given in Section 3.5.1.
  e) **Jess Time:** This is an internal Jess engine timestamp given to the fact at the moment of its appearance.
  f) **Type:** The Rule Engine categorizes the facts entries in the history according to their cause. Either one of three types can be assigned:
     - **Add:** The fact is added at this point in time.
     - **Remove:** The fact is removed at this point in time.
     - **Change:** The fact is changed. More precisely the value of at least one slot is changed.
  g) **Change:** The entries with the type *Change* have at least one slot where the value changed from one reasoning step to another. These changes are listed in this column as (slot value) pairs. The other types leave this column blank.

2. **Rule Trace:** This trace lists the rules fired during the reasoning procedure with each entry in the table symbolizing one specific rule firing. An entry consists out of four columns which have the following meaning:

   a) **Rule Name:** The unique identifier of the rule composed out of the module name and the rule name separated by two colons.
   b) **Pseudo Time:** This is an internal EM4J timestamp given to the rule at the moment of its firing. These times build the basis for generating the decision tree. A more detailed presentation of this timeline architecture has been given in Section 3.5.1.
   c) **Jess Time:** This is an internal Jess engine timestamp given to the rule at the moment of its firing.
   d) **Effect:** Rule firings can have a different impact depending on the circumstance of the firing. In principle there are two types of rule firings. The effect describes the impact that the rule's actions have on the factbase:
      - **Changing Firing:** The rule is fired (in other words its actions are executed) and actually change at least one slot value of a fact. In other words the factbase is changed.
      - **Redundant Firing:** The rule is fired. However the result of them is already present in the factbase. In other words the factbase is not changed, the firing of the rule was redundant. For example a rule is repeatedly fired then the first firing is of type changing. The repeated ones are redundant since the results of the rule's actions are already present in the factbase. But this is only true if the actions do not contain global variables or function respectively program calls.

3. **Decision Tree:** This graphical component shows the reasoning process. It presents the pathway of facts and rules leading to the conclusion. A detail description will be provided below in Section 4.4.2.1.



Fact Name	F...	Slots	Pseudo Ti...	Jess Ti...	Type	Chan
MAIN::IS_GEO	0	vegetation forest;gra...	0:1	0	ADD	
MAIN::IS_GEO	0	vegetation forest;gra...	0:2	2	CHANGE	subsoil malle
MAIN::IS_GEO	0	vegetation forest;gra...	0:3	4	CHANGE	danger_fine_
MAIN::IS_GEO	0	vegetation forest;gra...	0:4	6	CHANGE	bedding solid
MAIN::IS_GEO	0	vegetation forest;gra...	0:5	8	CHANGE	danger_recer
MAIN::IS_GEO	0	vegetation forest;gra...	0:6	10	CHANGE	danger_bedd
MAIN::IS_GEO	0	vegetation forest;gra...	0:7	12	CHANGE	danger_slope
MAIN::IS_GEO	0	vegetation forest;gra...	0:8	14	CHANGE	risk_defined Y
MAIN::IS_GEO	0	vegetation forest;gra...	0:8	16	CHANGE	risk_defined Y

Rule Name	Pseudo Time	Jess Time	Effect
MAIN::subsoil_mall...	0:1	1	Changing
MAIN::danger_fine...	0:2	3	Changing
MAIN::danger_fine...	0:3	5	Redundant
MAIN::bedding_no	0:3	5	Changing
MAIN::danger_fine...	0:4	7	Redundant
MAIN::bedding_no	0:4	7	Redundant
MAIN::subsoil_mall...	0:4	7	Redundant
MAIN::recent_defor...	0:4	7	Changing
MAIN::danger_fine...	0:5	9	Redundant
MAIN::bedding_no	0:5	9	Redundant
MAIN::subsoil_mall...	0:5	9	Redundant
MAIN::recent_defor...	0:5	9	Redundant
MAIN::bedding_low...	0:5	9	Changing
MAIN::danger_fine...	0:6	11	Redundant
MAIN::bedding_no	0:6	11	Redundant
MAIN::subsoil_mall...	0:6	11	Redundant
MAIN::recent_defor...	0:6	11	Redundant
MAIN::bedding_low...	0:6	11	Redundant
MAIN::recent_defor...	0:6	11	Redundant
MAIN::slope_angle...	0:6	11	Changing
MAIN::danger_fine...	0:7	13	Redundant
MAIN::bedding_no	0:7	13	Redundant
MAIN::subsoil_mall...	0:7	13	Redundant

Figure 4.11: EM4J - Decision tree view

4.4.2.1 Decision Tree

The graphical reasoning trace represents the main explanation tool of the Decision Tree Analyser. This graphical tree accommodates a list of functions and options providing the user with the possibility to comprehend the KBS and its reasoning that is contained in the archive:

- **Reasoning Trace:** The tree shows how the system came to its conclusion. With the help of a graphical representation the pathway through the clutter of information coded into the knowledge base can be investigated. The conditions leading to the firing of a rule as well as the actions performed thereafter can be seen in each step of the solution process.

- **Correlation/Implication:** The tree shows how the facts and rules work together forming the complete reasoning trace concluding for example in the system's computed risk factor.

- **Additional Information:** The tree contains additional information about the rules and facts. By hovering over a node a tooltip is shown. There are two possible versions:
  - Representing a rule: The left hand side and right hand side are shown
  - Representing a fact: The (slot value) pairs and, if available, the changed ones of them are shown.
- **Effect:** Since there are two types of rule firings these can be visually distinguished:
  - **Changing Firing ($\longrightarrow$):** The continuous line symbolizes that the fired rules have a direct effect on the factbase. In most cases these rules have **not** been fired in the steps before.
  - **Redundant Firing ($\dashrightarrow$):** The dashed line symbolizes that the fired rules have no direct effect on the factbase. In most cases these rules have been fired immediately the step before.

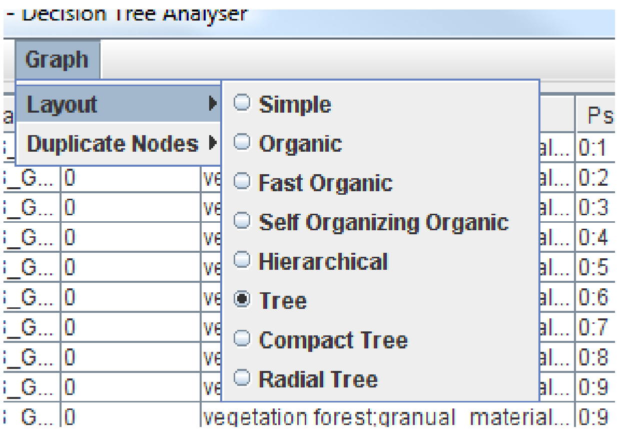
4.4.2.2 Interaction

After outlining the three graphical main parts of the Decision Tree Analyser the interaction between them is now in the focus. The rules respectively facts shown in the rule trace table respectively fact history table are connected to the corresponding nodes in the graphical tree. By selecting a node in the tree the corresponding rule entry respectively fact entry is highlighted. This is also true vice versa. Selecting an entry in one of the two tables results in highlighting the corresponding node.

4.4.2.3 Tree Transformation

The analyser possesses functions to alter the layout of the tree. There are several layout managers present in the system. By selecting the menu item *Graph/Layout* (see Figure 4.12) the list of possible layouts is shown:

- **Simple:** Implementation of a simple layout algorithm that places the nodes randomly on the screen.
- **Organic:** Implementation of a simulated annealing layout algorithm. This layout is based on [5].
- **Fast Organic:** Implementation of the Fruchterman-Reingold force-directed algorithm for node layout [7].
- **Self Organizing Organic:** Implementation of an inverted self-organizing maps layout based on Bernd Meyes's self-organizing maps methods [16].
- **Hierarchical:** Implementation of a layout placing the nodes according to their hierarchy.
- **Tree:** Implementation of a basic tree layout that places the nodes of a tree from north to south. If more than one tree is contained then these trees are placed from west to east.



Layout options: ○ Simple, ○ Organic, ○ Fast Organic, ○ Self Organizing Organic, ○ Hierarchical, ● Tree, ○ Compact Tree, ○ Radial Tree

Figure 4.12: *Graph/Layout* menu item

- **Compact Tree:** Implementation of a Moen layout. This layout focuses on placing the graph as compact as possible while still being nicely viewable [17].

- **Radial Tree:** Implementation of a radial tree layout. In this layout the root node is placed in the center with the children placed around it in concentric ovals.

4.4.2.4 Note Duplication

Large graphs respectively trees can be a little confusing because there are so many vertices and edges displayed. This problem can be divided into two subproblems. The first deals with the amount of nodes. A huge amount of nodes can lead to a confusing graph. However this problem can be diminished by spreading the graph. This is done automatically by EM4J. The second deals with the node degree. The degree of a node denotes the amount of incoming and outgoing edges of it. More precisely, the problem is that in general a high degree can lead to a confusing graph. This issue can be reduced through the *Graph/Duplicate Nodes* option (see Figure 4.13). This option duplicates nodes with a degree higher than the specified value.
The submenu opened when the *Graph/Duplicate Nodes* menu item is selected displays the following:

- **d = n (n= 5,10,20,50):** All the nodes with a degree greater than n are duplicated.

- **No:** In this case the duplication option is turned off.

This is a graphical duplication to assist the user's comprehension not to be confused with a real duplication of the fact in the rule engine!
Duplicated rules are marked with a "[D]" in the graphical representation at the end of their name.

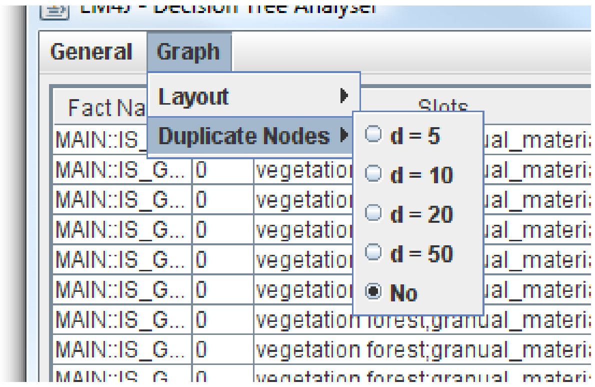


Figure 4.13: *Graph/Duplicate Nodes* menu item

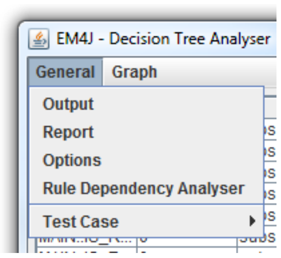


Figure 4.14: *General* menu item

4.4.2.5 Report Generation

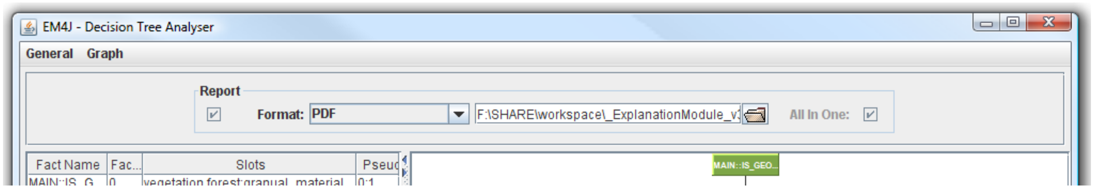
EM4J possesses the ability to generate reports from the archive shown in the Decision Tree Analyser. Every report contains a title page showing the fully qualified name of the archive. This title contains the full path on the file system including the name and the date of its generation. The report itself contains the initial facts, the output of the last rule, the fact history, the rule trace and the full text output of the fired rules.

The format of the report and its destination directory are chosen via the menu item *General/Options* (see Figure 4.14). Immediately after clicking this menu item an input window shows up in the Decision Tree Analyser (see Figure 4.15). The combo box on the left hand side lists the available report formats. Until now there is just the PDF[3] format supported. The text field on the right side next to the combo box shows the destination path of the generated report, followed by a button displaying the symbol of a document file. By pushing this button a file chooser is opened so that the user has the option to choose the destination and the name of the report to be produced. The default directory for reports is called "reports" and is located in the working directory of EM4J. The default name of the report has the following layout: "archive-name_TestCaseReport.pdf".

The actual report can be generated through the menu item *General/Report* (see Figure 4.14).

[3] `http://www.adobe.com/devnet/pdf/pdf_reference.html`

The system generates the report with a progress bar providing feedback. After the procedure is finished the PDF report is shown with the operation system's standard PDF viewer.



Format: PDF F:\SHARE\workspace_ExplanationModule_v1

All In One: ✓

Figure 4.15: Options from the Decision Tree Analyser

4.4.2.6 Text Output View

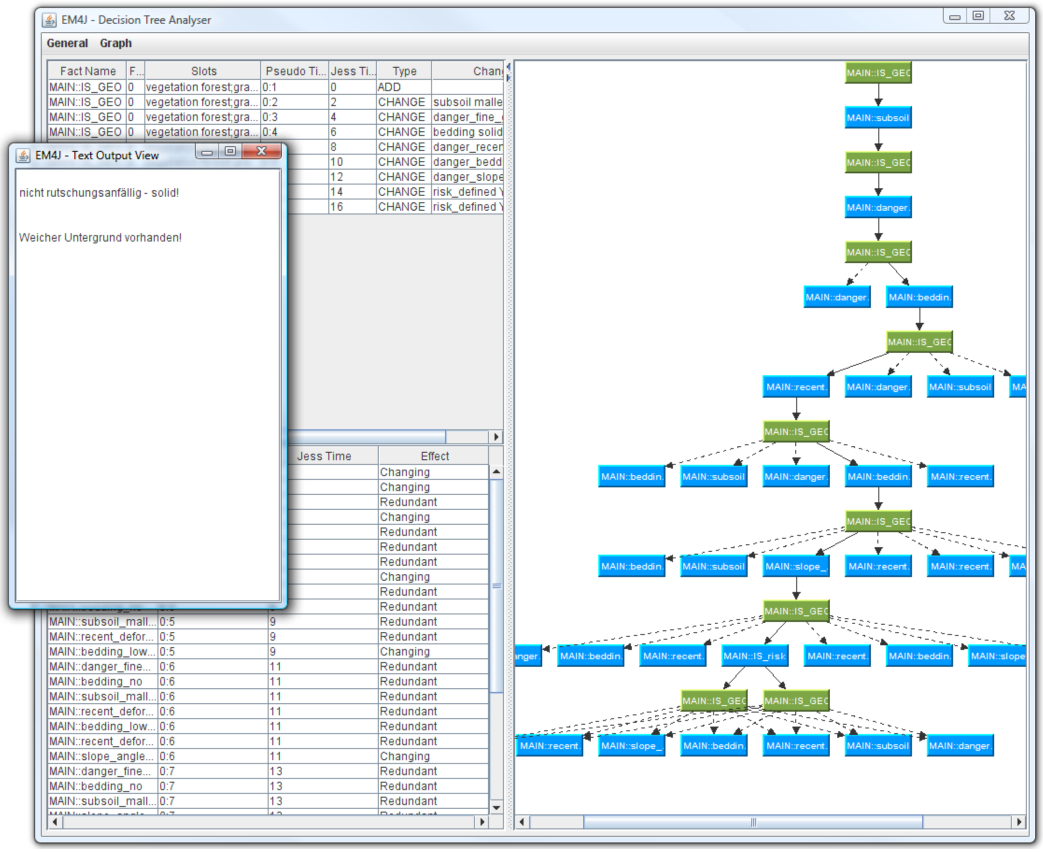
The Rule Engine possesses the ability to put output operations into the right hand side of rules. These operations print text on the standard output mechanism of the system, the console. Since the Rule Engine runs are decoupled from EM4J this text would be lost. Consequently the data must be redirected, written into the archive and is then in connection with the Decision Tree Analyser displayed in a separate window. This window is called Text Output View (see Figure 4.16). This view consists of a text area displaying the text generated by the rules separated through line breaks.
This separate window can be closed, minimized and maximized as usual. If it is closed it can be reopened via the menu item *General/Output* (see Figure 4.14).

4.4.3 Test Cases

From the structural point of view test cases are a part of the Decision Tree Analyser. However they constitute such a big part of the whole system that they are illustrated in detail. Test cases have the purpose to deepen, increase and extend the understanding respectively comprehension of user regarding the KBS and its behavior. In the current version of EM4J are two sorts of test cases implemented:

- **Simple Test Case:** The simple version of a test case is the one where the user is able to test the behavior of the system by choosing variations of the input characteristics. In other words the user is able to check the system on different input values. This is done through the menu item *General/Test Case/Simple Test Case.* There the user has the possibility to choose one of two options:

    1. **Empty Start State:** In this state the default values are set within each slot present in the initial facts (see Figure 4.17). Each template contains one such initial fact at the beginning.
    2. **Initial Start State:** In this state every slot has the values initially given as an input when the archive was created. The advantage of this feature is that the user has the initial state of the system at hand and can alter this state very easily instead of first



Fact Name	F...	Slots	Pseudo Ti...	Jess Ti...	Type	Chan
MAIN::IS_GEO	0	vegetation forest;gra...	0:1	0	ADD	
MAIN::IS_GEO	0	vegetation forest;gra...	0:2	2	CHANGE	subsoil malle
MAIN::IS_GEO	0	vegetation forest;gra...	0:3	4	CHANGE	danger_fine_
MAIN::IS_GEO	0	vegetation forest;gra...	0:4	6	CHANGE	bedding solid
				8	CHANGE	danger_recen
				10	CHANGE	danger_bedd
				12	CHANGE	danger_slope
				14	CHANGE	risk_defined
				16	CHANGE	risk_defined

```
nicht rutschungsanfällig - solid!

Weicher Untergrund vorhanden!
```

		Jess Time	Effect
			Changing
			Changing
			Redundant
			Changing
			Redundant
			Redundant
			Redundant
			Changing
			Redundant
			Redundant
MAIN::subsoil_mall...	0:5	9	Redundant
MAIN::recent_defor...	0:5	9	Redundant
MAIN::bedding_low...	0:5	9	Changing
MAIN::danger_fine...	0:6	11	Redundant
MAIN::bedding_no	0:6	11	Redundant
MAIN::subsoil_mall...	0:6	11	Redundant
MAIN::recent_defor...	0:6	11	Redundant
MAIN::bedding_low...	0:6	11	Redundant
MAIN::recent_defor...	0:6	11	Redundant
MAIN::slope_angle...	0:6	11	Changing
MAIN::danger_fine...	0:7	13	Redundant
MAIN::bedding_no	0:7	13	Redundant
MAIN::subsoil_mall...	0:7	13	Redundant

Figure 4.16: Decision Tree Analyser with Text Output View

selecting all the initial values (since there are only the template specific initial values set) and then changing them (see Figure 4.18). In other words the effort to introduce variations in the original setting is lowered.

- **Advanced Test Case:** The advanced test case is a more powerful tool. This version gives the user the opportunity to automatically test ranges of values of slots within the empty initial facts.

When the user generate facts for test cases he/she must be aware of the circumstance that empty slots can cause trouble during the execution of the test case!

4.4.3.1 Simple Test Case

Figure 4.21 shows the initial view of the Empty Start State Simple Test Case. At the top there is the factbase displayed. Below the same GUI as in the Decision Tree Analyser is shown with the same functionalities, options, menu items, etc., at hand.
With the tabs at the top the user has the possibility to choose between the different fact templates

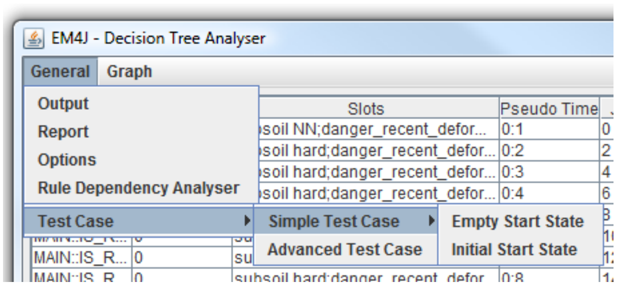


Figure 4.17: Simple Test Case menu item

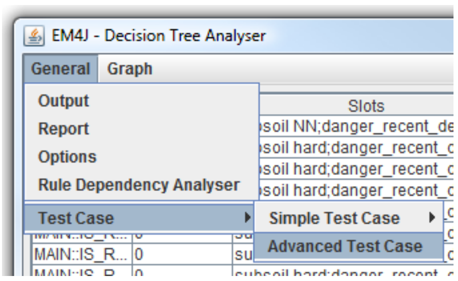


Figure 4.18: Advanced Test Case menu item

present in the system. A template describes the structure (e.g., values, slots, etc.) of the facts to be created with it. Each tab displays a table representing the initial fact defined with this template. The columns of each table are defined by the template it represents. Each table is filled in one of the following two ways:

1. **Empty Start State Simple Test Case:** The default configuration is one fact per template filled with its default values defined in the template.

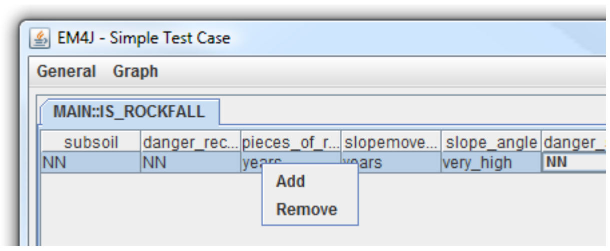
2. **Initial Start State Simple Test Case:** The tables are filled with the facts out of the initial configuration of the factbase saved in the archive.

The management of the initial facts is done by context menu interaction. There are two versions of the context menu:

1. There already exists at least one fact in the table and the context menu is opened by clicking with the right mouse button on one of the table entries (Figure 4.19). In this case the context menu shows two items, namely *Add* and *Remove* .

2. There is no fact listed or, although there exists at least one, the context menu is opened by clicking outside the table entries (see Figure 4.20). In this case the context menu has only one item, *Add* .

By selecting *Add* a new fact is created with the default values defined by the template. By selecting remove the selected fact is deleted from this test case.



subsoil	danger_rec...	pieces_of_r...	slopemove...	slope_angle	danger_
NN	NN	years	years	very_high	NN

Figure 4.19: Context menu of the Simple Test Case facts table

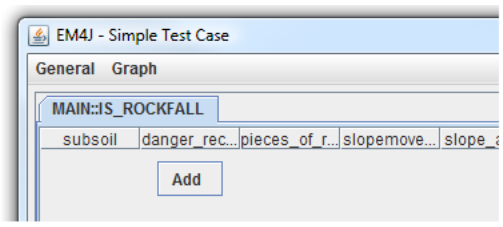
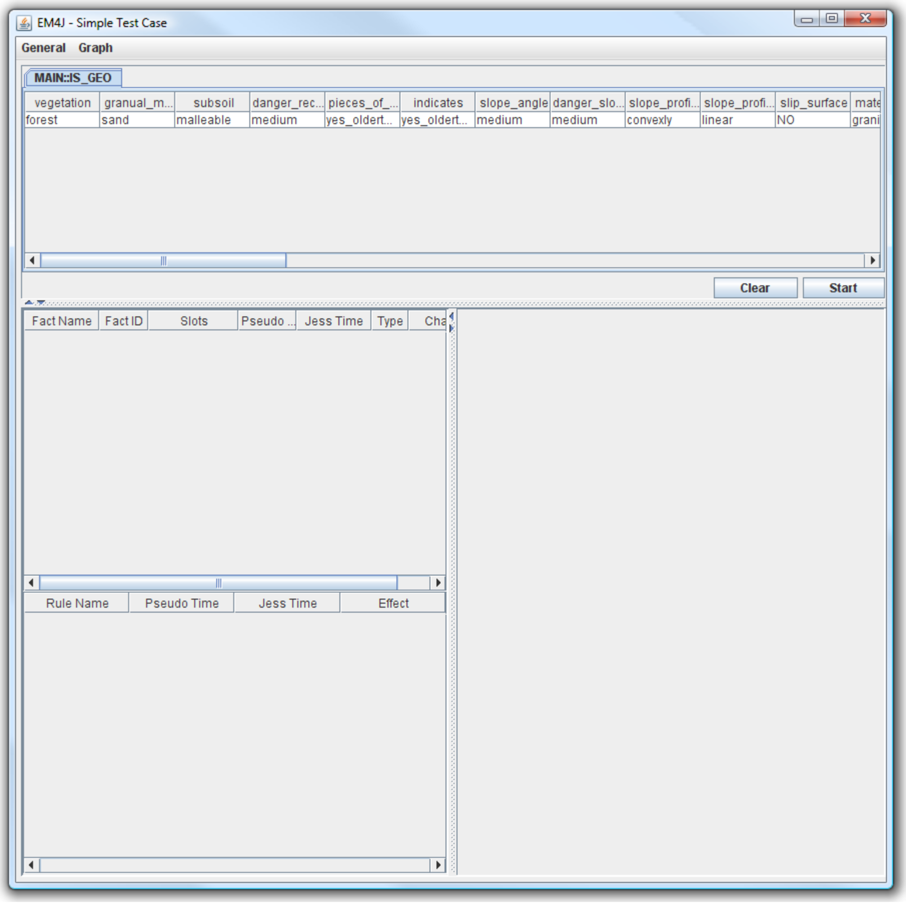


Figure 4.20: Context menu of the Simple Test Case facts table

The values of each slot can be changed by clicking with the left mouse button into the corresponding cell in the table. This either activates a combo box showing predefined values of this slot or an editable textfield for entering values. These predefined values are specified by the corresponding underlying template of the fact. Note that the system does not validate the user's input at this point!
Pushing the button *Start* initiates the Rule Engine that executes the program with the values selected above. These values are parsed into the respective facts and loaded into the factbase. To inform the user of the process a progress bar is shown immediately afterwards. The result, the decision tree and the textual output, are then given in the lower part of the Simple Test Case GUI respectively the Text Output View. Additionally, if the archive option is active, an archive of this test case is created. This is indicated by a feedback message displaying the name of the created archive (see Figure 4.22). This archive creation option can be customized through the option panel (see Figure 4.23). The panel is visualized by selecting the menu item *General/Options*. Here the user can activate or deactivate the archive creation by selecting or de-selecting the check box at the beginning. The text field with the file chooser next to it gives the user the capability to change the target directory of the test case archive to be created.
Pushing the button *Clear* resets the engine so that the user can alter the initial values and restart the rule engine.

4.4.3.2 Advanced Test Case

By opening this view (see Figure 4.25) the user has the possibility to test ranges of slot values. Every template in the system is represented by a tab at the top of the view. The following list describes the options displayed at the top of the template above the slots:

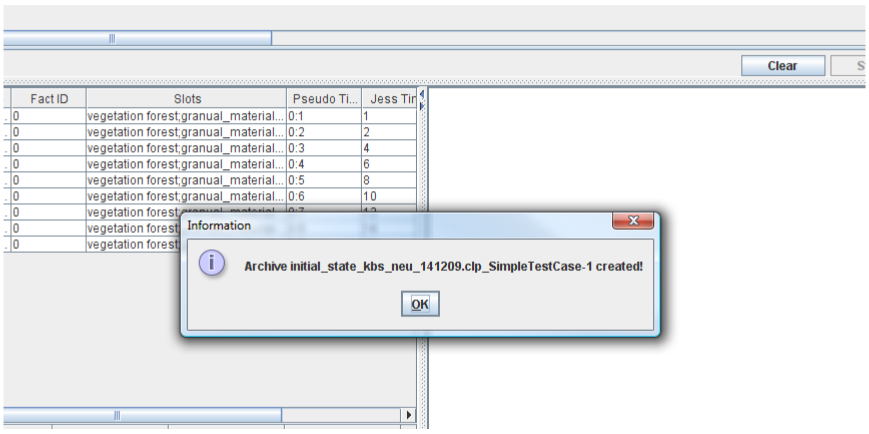


vegetation	granual_m...	subsoil	danger_rec...	pieces_of_...	indicates	slope_angle	danger_slo...	slope_profi...	slope_profi...	slip_surface	mate
forest	sand	malleable	medium	yes_oldert...	yes_oldert...	medium	medium	convexly	linear	NO	grani

Fact Name	Fact ID	Slots	Pseudo ...	Jess Time	Type	Cha

Rule Name	Pseudo Time	Jess Time	Effect

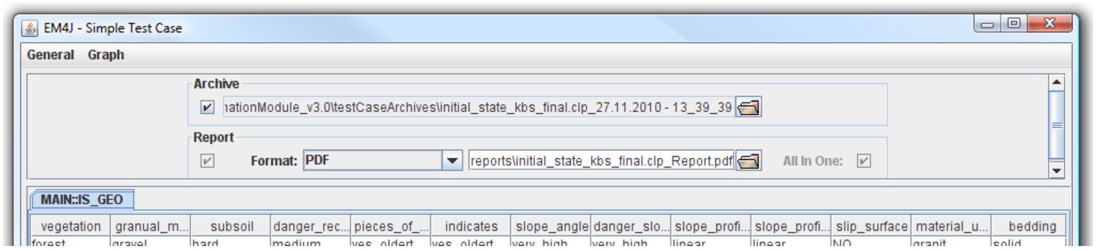
Figure 4.21: View of the simple test case



Fact ID	Slots	Pseudo Ti...	Jess Tir
0	vegetation forest;granual_material...	0:1	1
0	vegetation forest;granual_material...	0:2	2
0	vegetation forest;granual_material...	0:3	4
0	vegetation forest;granual_material...	0:4	6
0	vegetation forest;granual_material...	0:5	8
0	vegetation forest;granual_material...	0:6	10
0	vegetation forest;granual_material...	0:7	12
0	vegetation forest;		
0	vegetation forest;		

Archive initial_state_kbs_neu_141209.clp_SimpleTestCase-1 created!

Figure 4.22: Archive creation feedback message of the simple test case


EM4J - Simple Test Case

General Graph

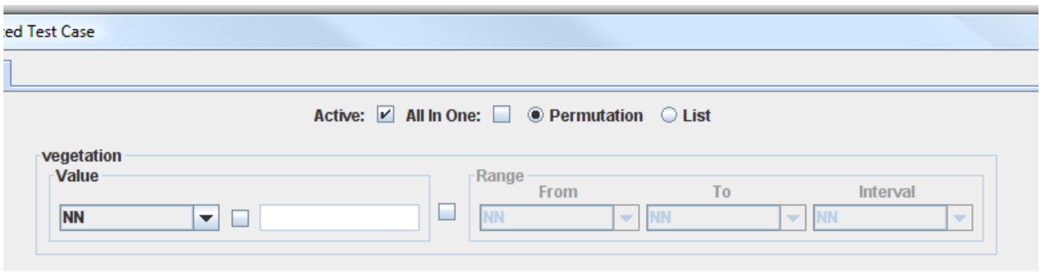
Archive: ☑ ıationModule_v3.0\testCaseArchives\initial_state_kbs_final.clp_27.11.2010 - 13_39_39

Report: ☑ Format: PDF reports\initial_state_kbs_final.clp_Report.pdf All In One: ☑

MAIN::IS_GEO

vegetation	granual_m...	subsoil	danger_rec...	pieces_of_...	indicates	slope_angle	danger_slo...	slope_profi...	slope_profi...	slip_surface	material_u...	bedding
forest	gravel	hard	medium	yes_oldert	yes_oldert	very_high	very_high	linear	linear	NO	granit	solid

Figure 4.23: Options view of the simple test case


ed Test Case

Active: ☑ All In One: ☐ [✓ Permutation] List

vegetation

Value: NN ☐

☐ Range: From NN To NN Interval NN

Figure 4.24: Enabling the template for fact creation

- **Active:** To create facts out of a specific template this check box has to be selected (see Figure 4.24). This enables the slots in the view.
- **All In One:** If this check box is selected then all the defined ranges and values are put into one single run of a job. This option is useful to create several facts of one template that should be included into one single execution.
- **Permutation/List:** Either one of these radio buttons can to be selected. The permutation option implies that the different slot ranges have a n-to-n relationship. This means that every value from the first range is combined with every value of the second range. For example, if range one has two values and range two has two values four facts are going to be created.
  The list option is bound to the condition that all specified ranges have to have the same amount of values. The reason for this is the fact creation process used for this option. This process does not create a permutation. Instead a simple list is generated. The number of entries is determined by the common cardinality of the ranges. The cardinality describes the number of values that a range contains. The actual generation process works as follows. The first value of each defined range is used for the first fact to be created, the second value of the range for the second fact, and so on. If these ranges do not have the same cardinality then there would be facts where not all slots are set.

4.4.3.2.1 Job Definition

A job is a bucket of runs. A run is another bucket of facts. Each run contains at least one

fact. These facts in the runs respectively jobs represent the combinations of values and ranges specified by the user. This is done by creating lists respectively permutations of the defined ranges that are either in one run (with the option "All In One" being set) or in separate runs.
The button *Add Fact(s)* is responsible for creating the configurations defined by the user and adding them to the processing queue, but not running them. The user has still the opportunity to add other configurations to be tested. By clicking the *Start* button the various jobs are going to be executed.

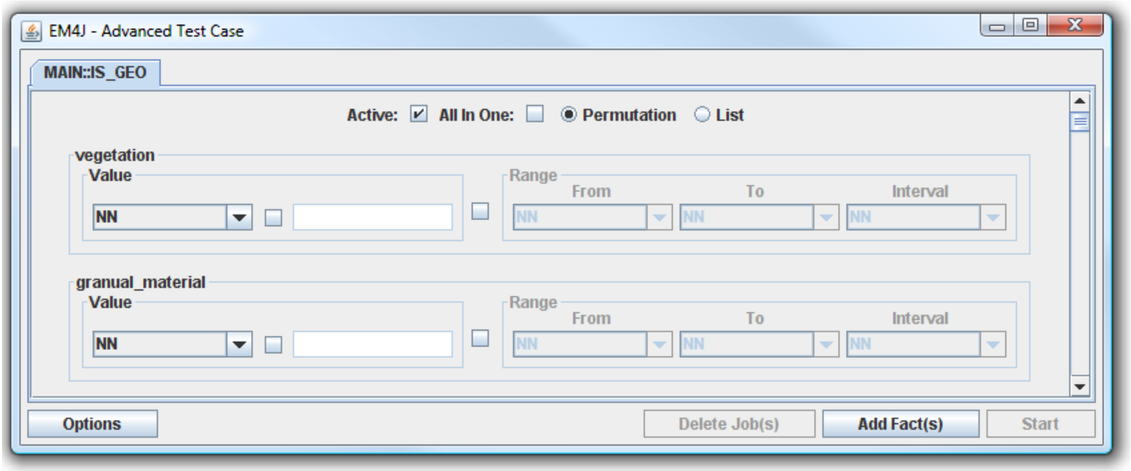


Figure 4.25: View of the advanced test case

4.4.3.2.2 Range and Value Definition

Each tab displays the slots defined by the template. Each slot constitutes a frame with the name of the slot on the top right side. These frames represent the configurations possible for each slot. Figure 4.26 shows the two versions of this graphical component.
The user must be aware of the fact that the system does not validate any slot input at this point! Inside of such a frame the user has the ability to either chose between selecting a distinct value or a value range for the slot. On the left hand side there is another frame where the user can specify a distinct value for the slot. On the right hand side there is also a frame where the user can specify a range for the slot.
At first each range configuration is disabled. In Figure 4.26, it can be seen that the range frame with its graphical components (text boxes, combo boxes, etc.) is disabled (painted in gray). By selecting the checkbox between the value and the range frame the range configuration is enabled. In turn the value configuration is disabled. This feature is implemented in a way that just either one of them can be used at the time but not both! This prevents input mistakes like defining both types for a slot. This case is further illustrated in Figure 4.27 which displays the version where the range configuration is enabled via the checkbox.
If a slot is displayed with combo boxes then the template provides pre-defined values for this slot. These values are the only ones that can be used within this slot. Otherwise a text field

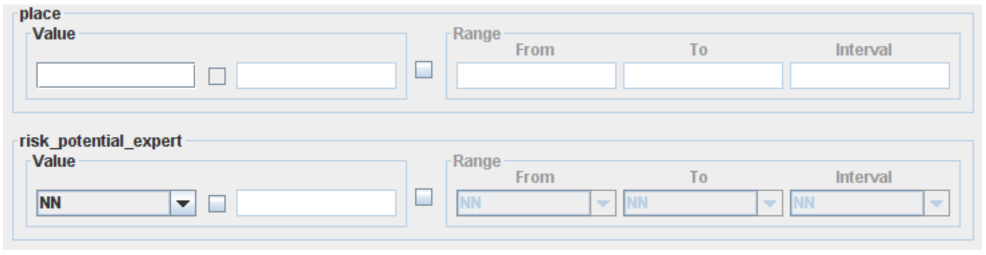


Figure 4.26: View of slots with text fields and combo boxes. The value creation frame is enabled

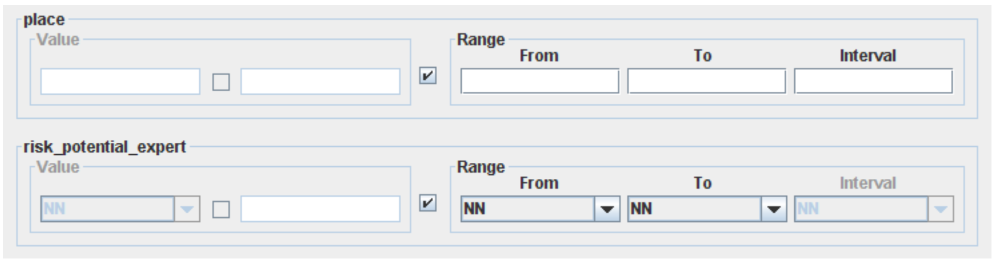


Figure 4.27: View of slots with text fields and combo boxes. The range creation frame is enabled

is given (the left text field is denoted) where the user can set the desired value(s). Figure 4.26 exemplifies both versions.
Additionally these text fields have another useful feature. It is possible to specify a list of values, which is also a range. However the user has more flexibility in defining this range since he/she does not have to specify a strict interval. These values in this list have to be separated by semicolons. An example is shown in Figure 4.28.

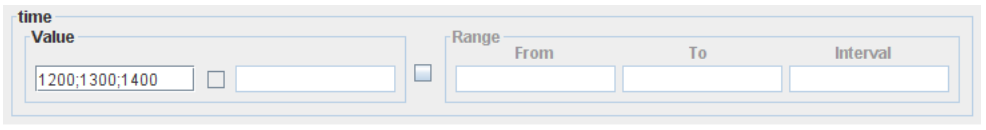
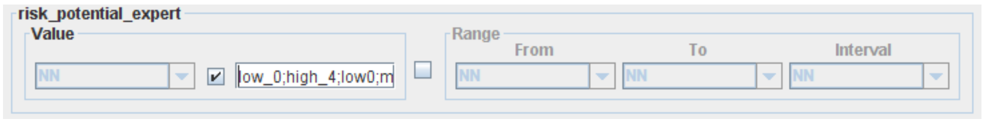


Figure 4.28: View of a slot with text fields. Input values are specified as a list

A special case is the right text field within the value frame. This text field is enabled whenever values are predefined for a slot. The sole purpose of this text field is to provide flexibility. The user gains the capability to specify a list of values for this slot (again separated by semicolons). This mirrors the circumstance when no values are specified by the template. The reason for this feature is to give the user the ability to define a range of values that is not bound to a specific interval. However the user is only allowed to input the predefined values! For better understanding Figure 4.29 shows an example.
To activate this special text field the check box between the two GUI items of the value frame has to be selected. The range configuration for a slot is displayed in one of two ways:



risk_potential_expert

Value: NN ✓ low_0;high_4;low0;m

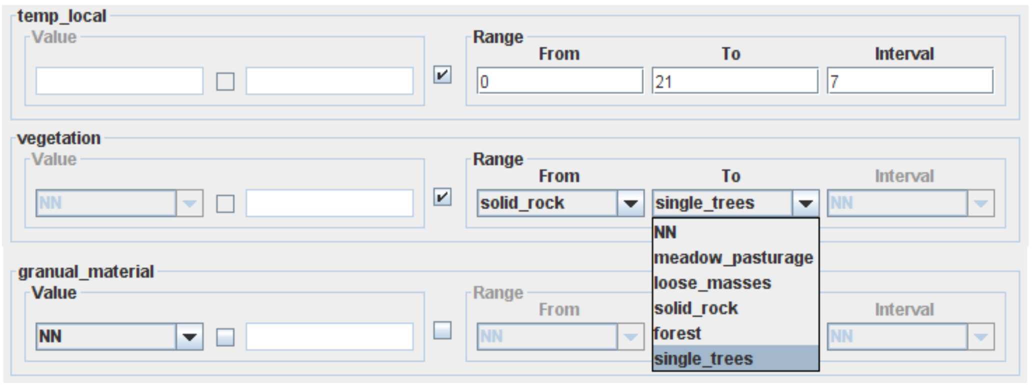
Range: From NN To NN Interval NN

Figure 4.29: View of a slot with predefined values. Input values are specified as a list

1. **No Predefined Values[4]:** In this case three text fields are given. All of them are enabled. The last text field defines the interval step size between the start and the end value.

2. **Predefined Values:** In this case there are three combo boxes given. The last combo box is disabled. This means that every value between the start and the end value is used. Floating point numbers are not a factor here since the list of predefined values is specified in the template by the developer. No interval steps can be defined. The order of the values is specified by the list that is dropped down by opening the combo box.

These two range definitions are shown in Figure 4.30. If the user pushes the *Add Fact(s)* button the actual setting is taken to create the facts. The actual structure how these created facts are put into runs of a new job depends on the generation options that are selected at the top of the window (see Figure 4.24). We already depicted them.



temp_local

Value

Range ✓ From 0 To 21 Interval 7

vegetation

Value NN

Range ✓ From solid_rock To single_trees Interval NN

NN
meadow_pasturage
loose_masses
solid_rock
forest
single_trees

granual_material

Value NN

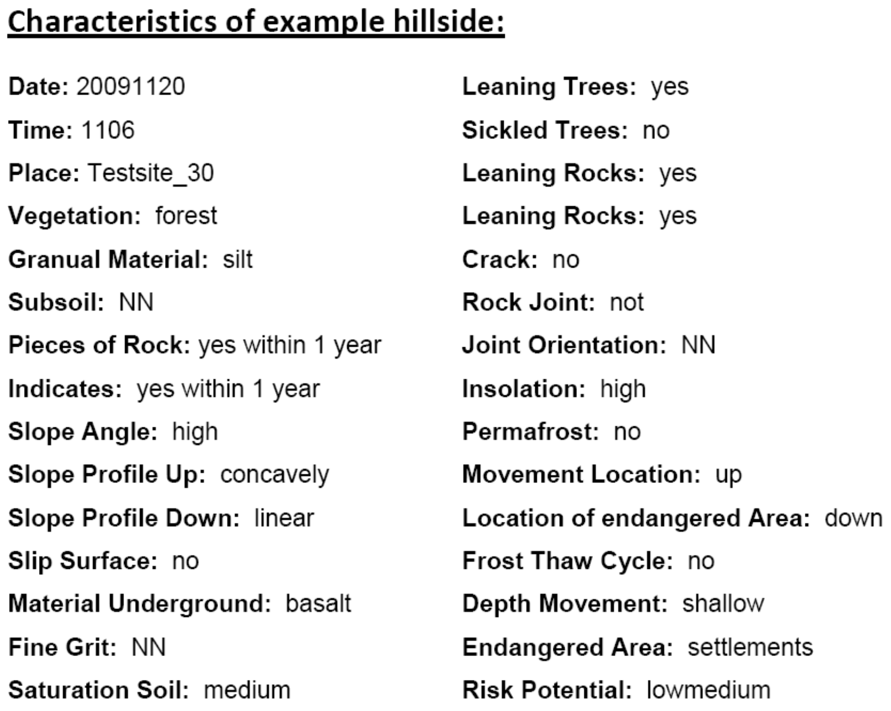
Range From NN Interval NN

Figure 4.30: View of slots with/-out predefined values. In both cases a range is specified

In Section 3.2.1.1 we already discussed the computation of the initial risk factor. Geologists fill out questionnaires to assess the characteristics of a geological site (e.g., a hillside). This is taken as an input for the initial risk factor generation.
The basic issue concerning this assessment is the fact that geologists lean towards variations. In other words characteristics are classified different (e.g., the slope angle being high or very high). The variations alone do not represent the problem itself. In fact it is the circumstance that although there are variations present geologists tend to the same risk classification. The system has to deal with these variations. In other words the system must be able to calculate appropriate

[4]In this case the data type restricts the amount of possible input values.

risk factors considering possible variations.
Additionally we want to stress the factor that especially during the development of a KBS the Advanced Test Case feature is of importance. It enables to identify errors early on instead of later during the KBS's application when possibly human lives are on the line. In order to give an example consider a false risk factor that indicates safety while a landslide happens.
To build on the example in Section 3.2.1.1 the filled questionnaire can be found in Figure 4.31. Investigation on the behavior of the system under different assessments for the same hillside is performed with the help of the Advanced Test Case feature of EM4J. Let us assume that there are several characteristics of this hillside where we encountered different expert opinions. These range from the slope angle being very high instead of high to the saturation of the soil being medium or high. Although there are variations in the assessment, the final risk categorization performed by experts is in line with each other. The system needs to cope with this circumstance. Therefore to investigate the behavior of InGES an Advanced Test Case is created.



Characteristics of example hillside:

Date:	20091120	**Leaning Trees:**	yes
Time:	1106	**Sickled Trees:**	no
Place:	Testsite_30	**Leaning Rocks:**	yes
Vegetation:	forest	**Leaning Rocks:**	yes
Granual Material:	silt	**Crack:**	no
Subsoil:	NN	**Rock Joint:**	not
Pieces of Rock:	yes within 1 year	**Joint Orientation:**	NN
Indicates:	yes within 1 year	**Insolation:**	high
Slope Angle:	high	**Permafrost:**	no
Slope Profile Up:	concavely	**Movement Location:**	up
Slope Profile Down:	linear	**Location of endangered Area:**	down
Slip Surface:	no	**Frost Thaw Cycle:**	no
Material Underground:	basalt	**Depth Movement:**	shallow
Fine Grit:	NN	**Endangered Area:**	settlements
Saturation Soil:	medium	**Risk Potential:**	lowmedium

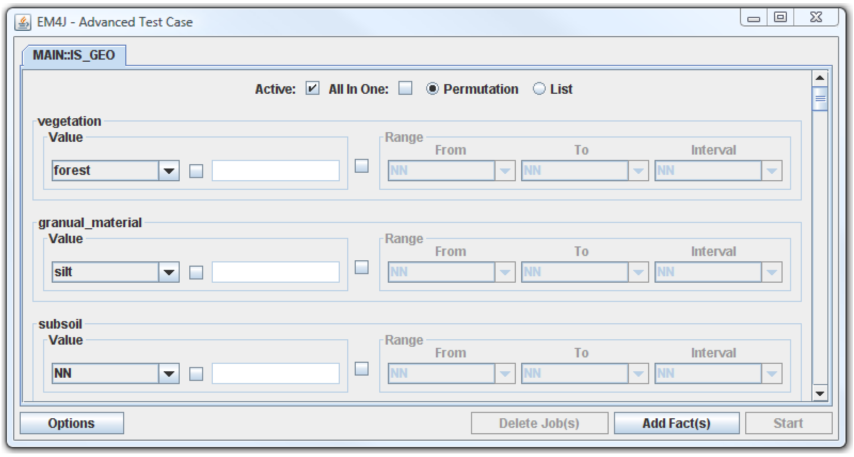
Figure 4.31: Example hillside (Legend: NN. . . slot not yet filled)

1. The questionnaire has to be filled into the corresponding tab of the Advanced Test Case window. The initial risk factor computation uses just one fact representing the questionnaire. Therefore we need to configure the settings as follows (see Figure 4.32 at the top):

   Active: Yes, since we need the only template to generate facts.

   All In One: No, since we need one generated fact per run not all generated facts into one run.

Permutation/List: Permutation is chosen since only one fact per execution is needed. This option generates one fact per run since the 'All In One' option is disabled. Otherwise all facts would be packed into one run respectively one factbase.


EM4J - Advanced Test Case

MAIN::IS_GEO

Active: ☑ All In One: ☐ ● Permutation ○ List

vegetation
Value: forest ☐ — Range: From NN, To NN, Interval NN ☐

granual_material
Value: silt ☐ — Range: From NN, To NN, Interval NN ☐

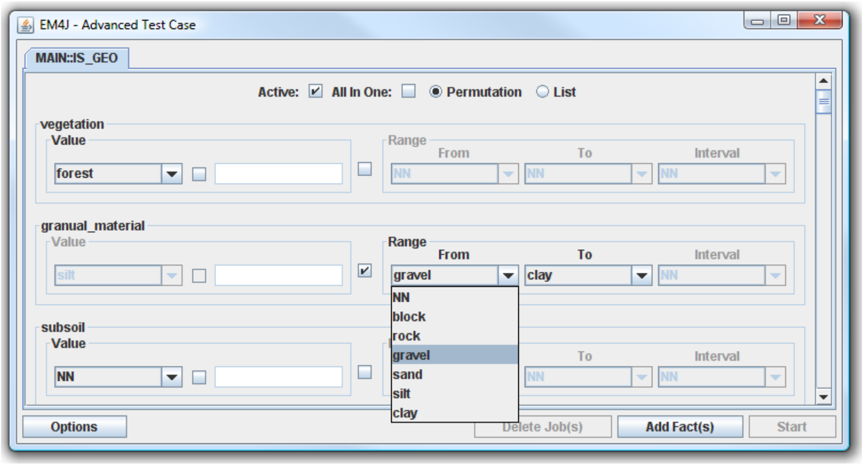
subsoil
Value: NN ☐ — Range: From NN, To NN, Interval NN ☐

Figure 4.32: Advanced Test Case example: step 1

2. Ranges have to be defined. This is done via the capabilities given by the Advanced Test Case input mask (see Figure 4.33). The user configures minor variations to test whether the computed initial risk factor changes. The user can even test all variations in order to assess the course of the risk factor computation. Especially inconsistencies like leaps in the assessment on minor variations are of interest. This indicates that there are possibly errors in the design of the KBS that cause these inconsistencies.
   The figure below shows a range definition. The slot *granual_material* is initially set to *silt*. However we want to test the impact of other possible categorizations. For example an other expert may categorize this slot as *sand* another as *clay*. The specific type of ranges to be tested are chosen by experts who are able to assess there importance. Note that not every variation is valid. Some can be combinations of values that just do not exist in nature.

3. Pressing the *Add Fact(s)* button generates the test setting. In other words the jobs to be executed are created. By clicking the *Options* button a table with the generated jobs is shown (see Figure 4.34). Details about these options are presented later on in this section. In Figure 4.34 it can be seen that there are four facts generated. Each fact is packed into one run respectively test case. This circumstance is shown in the second column by the checkbox that is not selected. The column '*Slots*' shows the configuration of each fact. Here the range defined in the previous step is mirrored in the facts. The range of slot *granual_material* is defined from *gravel* to *clay*. According to the list of allowed values shown in the figure above, the facts for this range are created.

4. Pressing the *Start* button executes this job.



Active: ☑ All In One: ☐ Permutation [✓] List

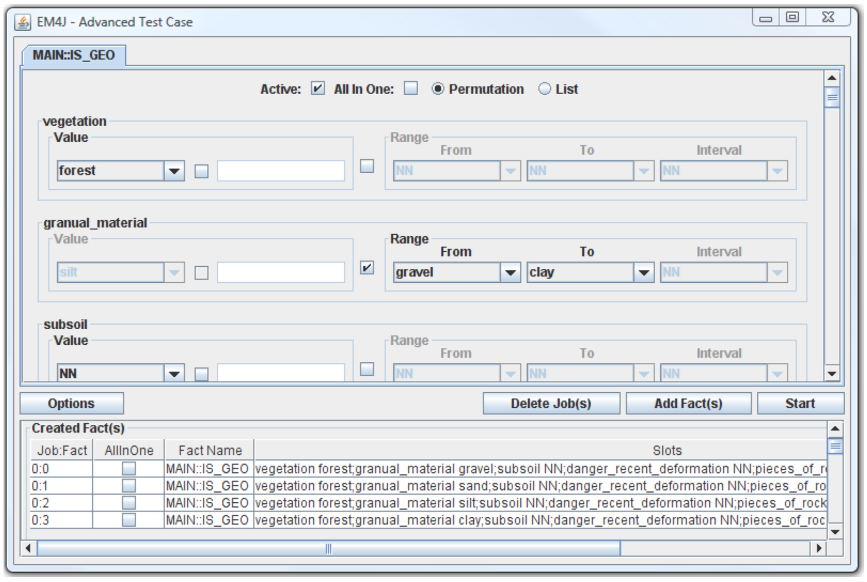
vegetation — Value: forest ☐ — Range ☐ — From: NN — To: NN — Interval: NN

granual_material — Value: silt ☐ — Range ☑ — From: gravel — To: clay — Interval: NN

From list: NN, block, rock, gravel, sand, silt, clay

subsoil — Value: NN ☐ — Range ☐ — To: NN — Interval: NN

Figure 4.33: Advanced Test Case example: step 2



Active: ☑ All In One: ☐ Permutation [✓] List

vegetation — Value: forest ☐ — Range ☐ — From: NN — To: NN — Interval: NN

granual_material — Value: silt ☐ — Range ☑ — From: gravel — To: clay — Interval: NN

subsoil — Value: NN ☐ — Range ☐ — From: NN — To: NN — Interval: NN

Job:Fact	AllInOne	Fact Name	Slots
0:0	☐	MAIN::IS_GEO	vegetation forest;granual_material gravel;subsoil NN;danger_recent_deformation NN;pieces_of_r
0:1	☐	MAIN::IS_GEO	vegetation forest;granual_material sand;subsoil NN;danger_recent_deformation NN;pieces_of_ro
0:2	☐	MAIN::IS_GEO	vegetation forest;granual_material silt;subsoil NN;danger_recent_deformation NN;pieces_of_rock
0:3	☐	MAIN::IS_GEO	vegetation forest;granual_material clay;subsoil NN;danger_recent_deformation NN;pieces_of_roc

Figure 4.34: Advanced Test Case example: step 3

This example shows a simple test case scenario with four instances respectively runs packed into one job. But these are not all instances since not even the slot 'granual_material' is fully exploited. By this it is meant that there are more allowed slot values available than used in this range. Depending on the amount of templates and their characteristics (number of slots, number of values per slot, etc.) the number of combinations respectively permutations is possibly unlimited.

4.4.3.2.3 Options

The *Options* button opens a pane (see Figure 4.35) that shows the following:

1. **Created Fact(s):** This table shows the already created jobs with all its representatives. The table has four columns — first the number of the job respectively the fact (separated by a colon), second if the facts of the job are all included into one run, third the name of the template and fourth the (slot value) pairs are given.

2. **Combine Facts Option:** If this option is set the facts created through clicking the *Add Fact(s)* button are added to the previously generated job. No new job is created!

3. **Save Job:** Saving the facts of a job in a file can be a nice asset to the user. For example, these facts can be used as input for Jess on another computer. The combo box on the left lists the supported file formats. In the current implementation of EM4J there are two formats available (Jess and CSV). The former results in a file that lists the facts in Jess syntax ready to be loaded into the Jess engine. The latter is a textual listing of the facts for further processing for example with Microsoft Excel[5].
   On the right side of the format combo box, the job to be saved can be chosen with another combo box. Last the button to actually save the job is given. By clicking the *Save* button a file chooser opens to set the filename as well as the path for the fact file to be saved. After the process is finished a feedback message is shown to the user.

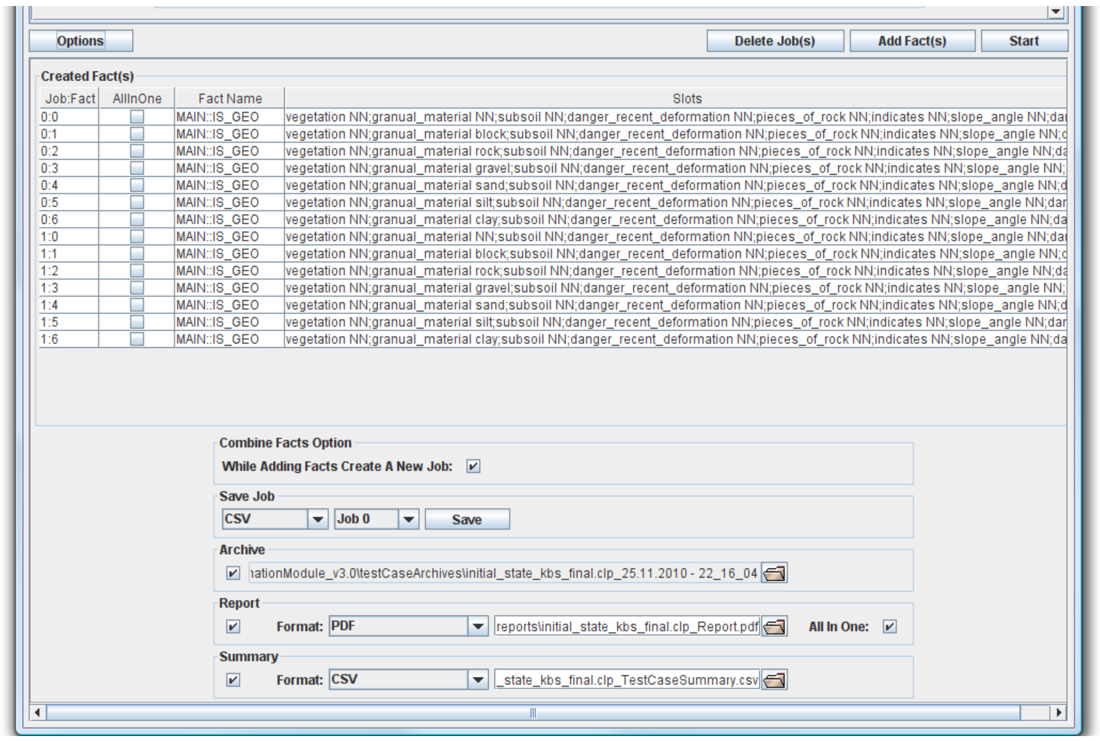
4. **Archive:** This option gives the user the ability to chose the destination directory where the archives of each run are going to be created. The checkbox on the left hand side enables respectively disables this option. The textfield on the right hand side displays the destination path with the file chooser button next to it where this path can be changed.

5. **Report:** This option gives the user the opportunity to change the configuration of the report generation. The checkbox on the left hand side enables respectively disables this feature. The combo box next to it gives the user the ability to change the format of the report. At this point only PDF reports are supported by the system. Then the report destination path, including the name of the report, is shown in the textfield. The folder button on the right side opens a file chooser so that the user can change the destination path. Last but not least on the outer right side there is another vital flag. The “All In One” flag is especially of importance if a huge amount of runs has to be processed. If this flag is selected then all the report data of the runs is collected in memory and afterwards written into one report file. Otherwise a report for every run is generated and therefore less memory is used during the execution. The activation of this flag can lead to run-time memory overflow errors!

6. **Summary:** This option gives the user the opportunity to change the configuration of the summary generation. A summary is compounded of entries that are dependent on the format. Each entry represents one run while containing the job respectively run number and the last rule fired by the engine.

[5]http://office.microsoft.com/de-at/excel

The checkbox on the left hand side enables respectively disables this option. The combo box next to it gives the user the ability to change the format of the summary. At this point only CSV summaries are supported by the system. Then the summary destination path, including the name of the summary file, is shown in the textfield. The folder button on the outer right side opens a file chooser so that the user can change the destination path.



Created Fact(s)

Job:Fact	AllInOne	Fact Name	Slots
0:0	☐	MAIN::IS_GEO	vegetation NN;granual_material NN;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;dar
0:1	☐	MAIN::IS_GEO	vegetation NN;granual_material block;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;c
0:2	☐	MAIN::IS_GEO	vegetation NN;granual_material rock;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;da
0:3	☐	MAIN::IS_GEO	vegetation NN;granual_material gravel;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;
0:4	☐	MAIN::IS_GEO	vegetation NN;granual_material sand;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;d
0:5	☐	MAIN::IS_GEO	vegetation NN;granual_material silt;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;dar
0:6	☐	MAIN::IS_GEO	vegetation NN;granual_material clay;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;da
1:0	☐	MAIN::IS_GEO	vegetation NN;granual_material NN;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;dar
1:1	☐	MAIN::IS_GEO	vegetation NN;granual_material block;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;c
1:2	☐	MAIN::IS_GEO	vegetation NN;granual_material rock;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;da
1:3	☐	MAIN::IS_GEO	vegetation NN;granual_material gravel;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;
1:4	☐	MAIN::IS_GEO	vegetation NN;granual_material sand;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;d
1:5	☐	MAIN::IS_GEO	vegetation NN;granual_material silt;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;dar
1:6	☐	MAIN::IS_GEO	vegetation NN;granual_material clay;subsoil NN;danger_recent_deformation NN;pieces_of_rock NN;indicates NN;slope_angle NN;da

Combine Facts Option

While Adding Facts Create A New Job: ✓

Save Job

CSV | Job 0

Archive

✓ ıationModule_v3.0\testCaseArchives\initial_state_kbs_final.clp_25.11.2010 - 22_16_04

Report

✓ Format: PDF | reports\initial_state_kbs_final.clp_Report.pdf | All In One: ✓

Summary

✓ Format: CSV | _state_kbs_final.clp_TestCaseSummary.csv

Figure 4.35: Visible options pane of the Advanced Test Case window

After finishing the creation of jobs the button *Start* has to be pushed. Now the processing of the jobs in the queue is launched. A progress bar is shown (see Figure 4.36) informing the user about the progress of the procedure. Immediately after finishing the work a message dialog shown in Figure 4.37 pops up that informs the user about the completion.

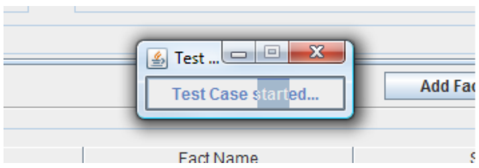


Figure 4.36: Progress bar shown during the job processing

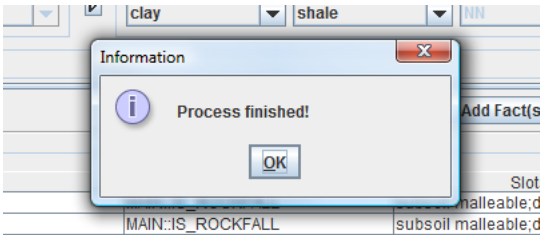


Figure 4.37: Message dialog indicating that the job processing is finished

4.5 Rule Dependency Analyser

The Rule Dependency Analyser is another tool within EM4J. Its purpose is to investigate the static dependencies between rules. It can either be started as a standalone application or as an investigation tool inside the Decision Tree Analyser.
First we describe the standalone version. An input window (see Figure 4.38) is displayed that gives the user the ability to choose the Jess program to investigate. This window displays two buttons and a textfield. The textfield shows the name of the opened file to be investigated. Right next to it the button *Open* opens a file chooser for selecting clp[6]-files . This is the file extension of Jess programs. After choosing a Jess program the system immediately loads the rules from this file. By clicking the button *Dependency Graph* another window is opened. This one is the actual analyzing window.

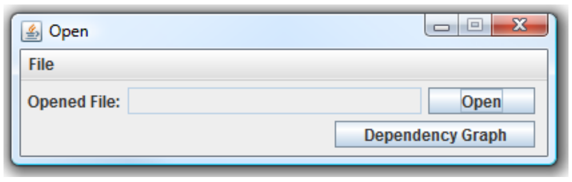
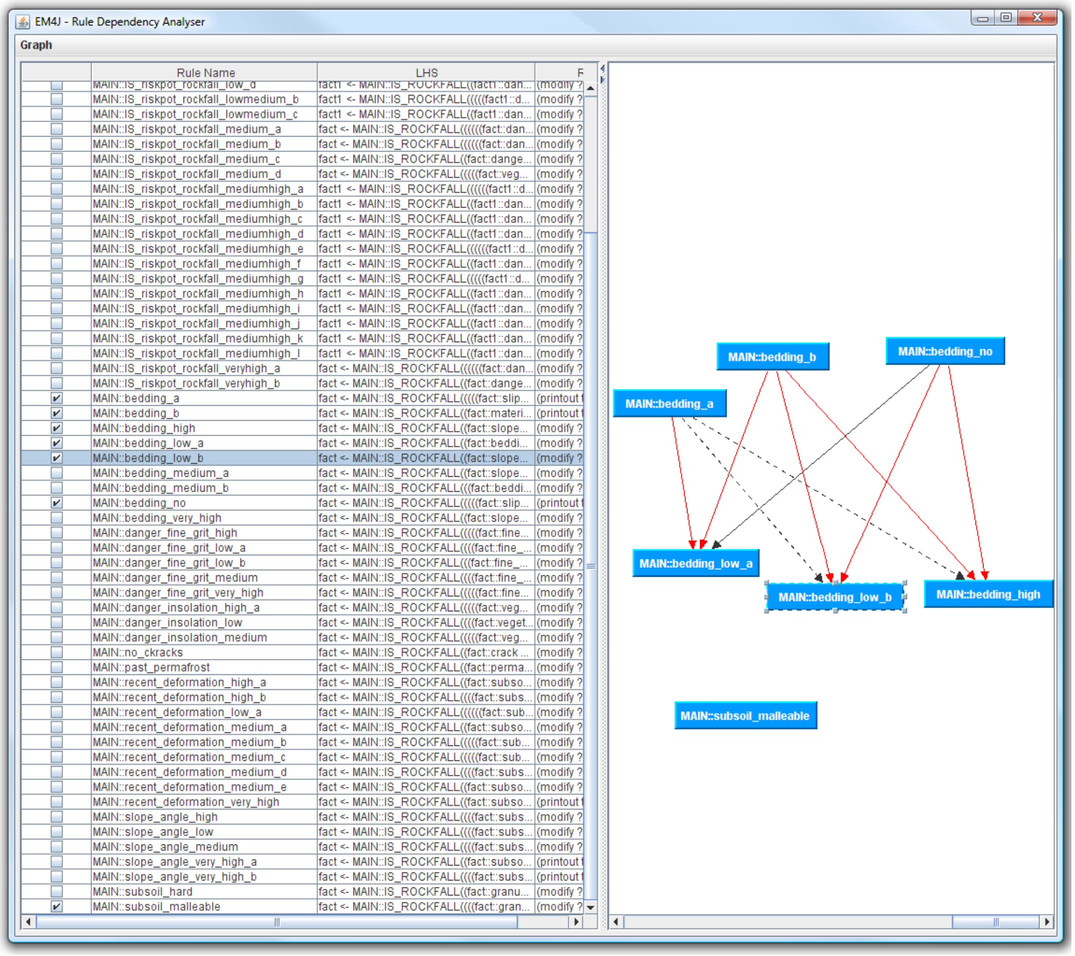


Figure 4.38: Rule Dependency Analyser initial window (standalone verion)

Second we describe how the Rule Dependency Analyser is started via the Decision Tree Analyser. In the Decision Tree View of the Decision Tree Analyser, the menu item *General/Rule Dependency Analyser* (see Figure 4.14) opens the Rule Dependency Analyser. The rules out of the archive already under investigation are loaded.
In both cases the opened window is now the actual working place (see Figure 4.39). It consists of two parts. On the left hand side there is a table containing all the rules of the system. On the right hand side there is the dependency graph shown.
The rule table has four columns. By the first one, the rule can be selected in order to investigate its dependencies to other rules. The second displays the name of the rule. The third respectively fourth column show the left hand side (LHS) or the preconditions respectively the ride hand side (RHS) or the actions of each rule.

[6]This file extension denotes Jess programs.



	Rule Name	LHS	F
☐	MAIN::IS_riskpot_rockfall_low_d	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_lowmedium_b	fact1 <- MAIN::IS_ROCKFALL(((((fact1::d...	(modify ?
☐	MAIN::IS_riskpot_rockfall_lowmedium_c	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_medium_a	fact <- MAIN::IS_ROCKFALL((((((fact::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_medium_b	fact <- MAIN::IS_ROCKFALL((((((fact::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_medium_c	fact <- MAIN::IS_ROCKFALL((fact::dange...	(modify ?
☐	MAIN::IS_riskpot_rockfall_medium_d	fact <- MAIN::IS_ROCKFALL(((((fact::veg...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_a	fact1 <- MAIN::IS_ROCKFALL((((((fact1::d...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_b	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_c	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_d	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_e	fact1 <- MAIN::IS_ROCKFALL((((((fact1::d...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_f	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_g	fact1 <- MAIN::IS_ROCKFALL(((((fact1::d...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_h	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_i	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_j	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_k	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_mediumhigh_l	fact1 <- MAIN::IS_ROCKFALL((fact1::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_veryhigh_a	fact <- MAIN::IS_ROCKFALL((((((fact::dan...	(modify ?
☐	MAIN::IS_riskpot_rockfall_veryhigh_b	fact <- MAIN::IS_ROCKFALL((fact::dange...	(modify ?
✓	MAIN::bedding_a	fact <- MAIN::IS_ROCKFALL(((((fact::slip...	(printout t
✓	MAIN::bedding_b	fact <- MAIN::IS_ROCKFALL((fact::materi...	(printout t
✓	MAIN::bedding_high	fact <- MAIN::IS_ROCKFALL((fact::slope...	(modify ?
✓	MAIN::bedding_low_a	fact <- MAIN::IS_ROCKFALL((fact::beddi...	(modify ?
✓	MAIN::bedding_low_b	fact <- MAIN::IS_ROCKFALL((fact::slope...	(modify ?
☐	MAIN::bedding_medium_a	fact <- MAIN::IS_ROCKFALL((fact::slope...	(modify ?
☐	MAIN::bedding_medium_b	fact <- MAIN::IS_ROCKFALL(((fact::beddi...	(modify ?
✓	MAIN::bedding_no	fact <- MAIN::IS_ROCKFALL(((((fact::slip...	(printout t
☐	MAIN::bedding_very_high	fact <- MAIN::IS_ROCKFALL((fact::slope...	(modify ?
☐	MAIN::danger_fine_grit_high	fact <- MAIN::IS_ROCKFALL(((((fact::fine...	(modify ?
☐	MAIN::danger_fine_grit_low_a	fact <- MAIN::IS_ROCKFALL((((fact::fine_...	(modify ?
☐	MAIN::danger_fine_grit_low_b	fact <- MAIN::IS_ROCKFALL(((fact::fine_...	(modify ?
☐	MAIN::danger_fine_grit_medium	fact <- MAIN::IS_ROCKFALL((((fact::fine_...	(modify ?
☐	MAIN::danger_fine_grit_very_high	fact <- MAIN::IS_ROCKFALL(((((fact::fine...	(modify ?
☐	MAIN::danger_insolation_high_a	fact <- MAIN::IS_ROCKFALL(((((fact::veg...	(modify ?
☐	MAIN::danger_insolation_low	fact <- MAIN::IS_ROCKFALL((((fact::veget...	(modify ?
☐	MAIN::danger_insolation_medium	fact <- MAIN::IS_ROCKFALL(((((fact::veg...	(modify ?
☐	MAIN::no_ckracks	fact <- MAIN::IS_ROCKFALL((fact::crack ...	(modify ?
☐	MAIN::past_permafrost	fact <- MAIN::IS_ROCKFALL((fact::perma...	(modify ?
☐	MAIN::recent_deformation_high_a	fact <- MAIN::IS_ROCKFALL((fact::subso...	(modify ?
☐	MAIN::recent_deformation_high_b	fact <- MAIN::IS_ROCKFALL((((fact::subs...	(modify ?
☐	MAIN::recent_deformation_low_a	fact <- MAIN::IS_ROCKFALL((((((fact::sub...	(modify ?
☐	MAIN::recent_deformation_medium_a	fact <- MAIN::IS_ROCKFALL((fact::subso...	(modify ?
☐	MAIN::recent_deformation_medium_b	fact <- MAIN::IS_ROCKFALL(((((fact::sub...	(modify ?
☐	MAIN::recent_deformation_medium_c	fact <- MAIN::IS_ROCKFALL(((((fact::sub...	(modify ?
☐	MAIN::recent_deformation_medium_d	fact <- MAIN::IS_ROCKFALL((((fact::subs...	(modify ?
☐	MAIN::recent_deformation_medium_e	fact <- MAIN::IS_ROCKFALL((fact::subso...	(modify ?
☐	MAIN::recent_deformation_very_high	fact <- MAIN::IS_ROCKFALL((fact::subso...	(printout t
☐	MAIN::slope_angle_high	fact <- MAIN::IS_ROCKFALL((((fact::subs...	(modify ?
☐	MAIN::slope_angle_low	fact <- MAIN::IS_ROCKFALL((((fact::subs...	(modify ?
☐	MAIN::slope_angle_medium	fact <- MAIN::IS_ROCKFALL((((fact::subs...	(modify ?
☐	MAIN::slope_angle_very_high_a	fact <- MAIN::IS_ROCKFALL((fact::subso...	(printout t
☐	MAIN::slope_angle_very_high_b	fact <- MAIN::IS_ROCKFALL((((fact::subs...	(printout t
☐	MAIN::subsoil_hard	fact <- MAIN::IS_ROCKFALL((fact::granu...	(modify ?
✓	MAIN::subsoil_malleable	fact <- MAIN::IS_ROCKFALL((((fact::gran...	(modify ?

Figure 4.39: View of the Rule Dependency Analyser

4.5.1 The Analyse Process and Graph Characteristics

The selection (e.g., the ticks in Figure 4.39) determines the shape of the graph. The actual rules selected are used for the dependency analysis. The graph is instantly updated whenever the selection is changed by the user. In this process the analyser building the graph looks at the left hand sides of the selected rules and checks the conditions in connection with the actions of all other selected rules. For example if the actions of rule `A` are executed the question is asked whether its actions (the RHS) meet any, some or all of the conditions of another rule `B`. Remember, we are strictly talking about a *static* analysis of dependencies that is based on the program code and *not* performed during run time. In other words no variables, function or program calls, etc. are evaluated.

The nodes displayed in the graph represent the rules selected. The arcs between them show the different types of dependencies. Actually the graph displays four types of dependencies (see Figure 4.40):

1. **No Arc:** No arc is drawn between two nodes. In this case the two rules have *no* dependency.
2. **Dashed Black Arc:** In this case the source node of the arc represents the rule where its actions meet at least one of the conditions of the other sink rule. There exists a *partial* dependency.
3. **Continuous Black Arc:** In this case the actions of the source rule meet *all* the conditions of the sink rule. There exists a *full* dependency between them.
4. **Continuous Red Arc:** In this case the actions of the source rule `A` *block* the sink rule `B`. In other words the actions of rule `A` dissatisfy at least one condition of rule `B` so that rule `B` cannot be fired. This holds until this blocking state is reversed. There exists a *negative* dependency.

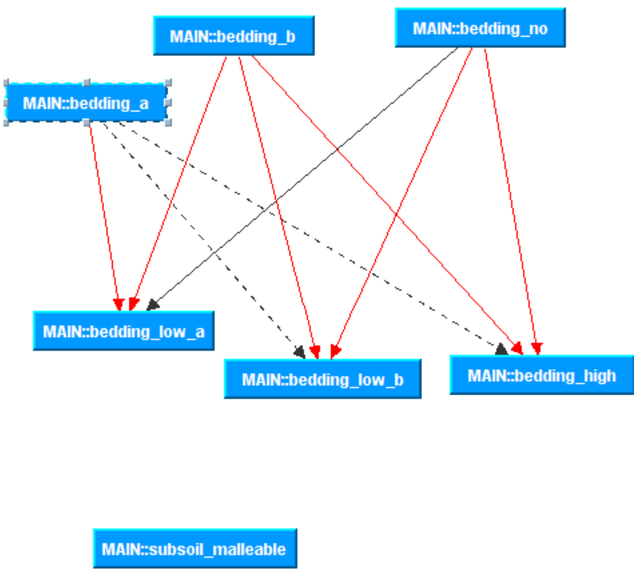
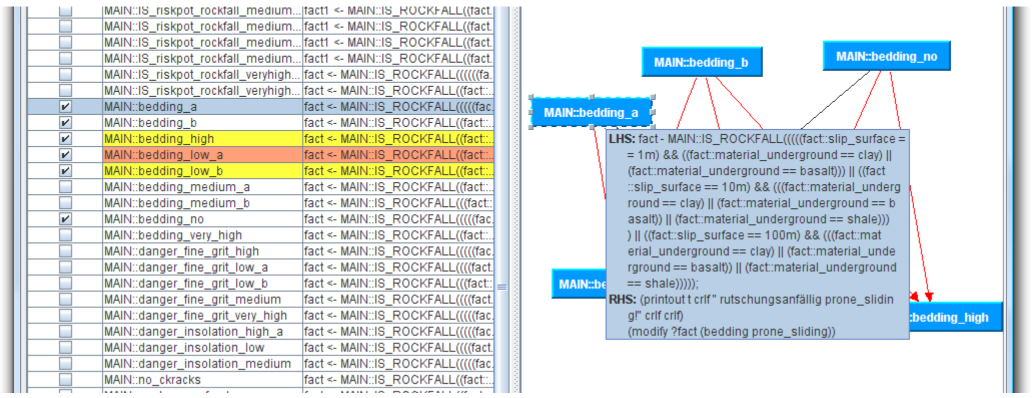


Figure 4.40: Exemplary rule dependency graph showing the different arc types

4.5.1.1 Additional Features

- Like within the Decision Tree Analyser the nodes possess the same tooltip capabilities (see Figure 4.41). If the mouse is moved on a rule its left hand side and right hand side are displayed.
- For easier operation the rule table possesses a context menu (see Figure 4.42) to select and deselect all the displayed rules.



☐	MAIN::IS_riskpot_rockfall_medium...	fact1 <- MAIN::IS_ROCKFALL((fact.
☐	MAIN::IS_riskpot_rockfall_medium...	fact1 <- MAIN::IS_ROCKFALL((fact.
☐	MAIN::IS_riskpot_rockfall_medium...	fact1 <- MAIN::IS_ROCKFALL((fact.
☐	MAIN::IS_riskpot_rockfall_medium...	fact1 <- MAIN::IS_ROCKFALL((fact.
☐	MAIN::IS_riskpot_rockfall_veryhigh...	fact <- MAIN::IS_ROCKFALL((((((fa.
☐	MAIN::IS_riskpot_rockfall_veryhigh...	fact <- MAIN::IS_ROCKFALL((fact::..
✓	MAIN::bedding_a	fact <- MAIN::IS_ROCKFALL(((((fac.
✓	MAIN::bedding_b	fact <- MAIN::IS_ROCKFALL((fact::..
✓	MAIN::bedding_high	fact <- MAIN::IS_ROCKFALL((fact::..
✓	MAIN::bedding_low_a	fact <- MAIN::IS_ROCKFALL((fact::..
✓	MAIN::bedding_low_b	fact <- MAIN::IS_ROCKFALL((fact::..
☐	MAIN::bedding_medium_a	fact <- MAIN::IS_ROCKFALL((fact::..
☐	MAIN::bedding_medium_b	fact <- MAIN::IS_ROCKFALL(((fact::
✓	MAIN::bedding_no	fact <- MAIN::IS_ROCKFALL(((((fac.
☐	MAIN::bedding_very_high	fact <- MAIN::IS_ROCKFALL((fact::..
☐	MAIN::danger_fine_grit_high	fact <- MAIN::IS_ROCKFALL(((((fac.
☐	MAIN::danger_fine_grit_low_a	fact <- MAIN::IS_ROCKFALL((((fact.
☐	MAIN::danger_fine_grit_low_b	fact <- MAIN::IS_ROCKFALL(((fact::
☐	MAIN::danger_fine_grit_medium	fact <- MAIN::IS_ROCKFALL((((fact.
☐	MAIN::danger_fine_grit_very_high	fact <- MAIN::IS_ROCKFALL(((((fac.
☐	MAIN::danger_insolation_high_a	fact <- MAIN::IS_ROCKFALL(((((fac.
☐	MAIN::danger_insolation_low	fact <- MAIN::IS_ROCKFALL((((fact.
☐	MAIN::danger_insolation_medium	fact <- MAIN::IS_ROCKFALL(((((fac.
☐	MAIN::no_ckracks	fact <- MAIN::IS_ROCKFALL((fact::..

```
LHS: fact - MAIN::IS_ROCKFALL(((((fact::slip_surface =
     = 1m) && ((fact::material_underground == clay) ||
     (fact::material_underground == basalt))) || ((fact
     ::slip_surface == 10m) && (((fact::material_underg
     round == clay) || (fact::material_underground == b
     asalt)) || (fact::material_underground == shale)))
     ) || ((fact::slip_surface == 100m) && (((fact::mat
     erial_underground == clay) || (fact::material_unde
     rground == basalt)) || (fact::material_underground
     == shale)))));
RHS: (printout t crlf " rutschungsanfällig prone_slidin
     g!" crlf crlf)
     (modify ?fact (bedding prone_sliding))
```

Figure 4.41: Dependency highlighting and tooltip capabilities

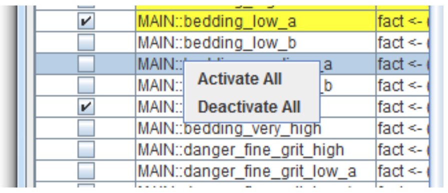


Figure 4.42: Context menu of the rule table

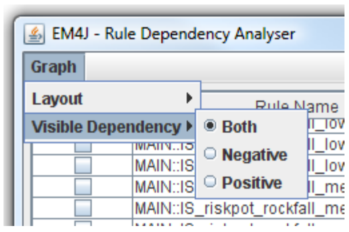


Figure 4.43: Visible dependency menu

- An arc filter to change the visibility of the different arc types is implemented. Through the menu item *Graph/Visible Dependency* (see Figure 4.43) the user can choose between three options:
  1. **Both:** All arc types of the graph are shown.
  2. **Negative:** Only the continuous red arcs respectively *negative* dependencies are shown.
  3. **Positive:** Only the *positive* dependencies are shown. These are the continuous (*full* dependency) and the dashed black arcs (*partial* dependency).
- By selecting a rule in the graph or an entry in the table (which must be selected via the checkbox) its dependencies to other rules are shown in the table. This is done by highlighting the rules it is affecting (see Figure 4.41). There exist two highlighting colors:
  1. **Light Yellow:** These rules have a *positive* dependency to the selected rule.
  2. **Light Salmon:** These rules have a *negative* dependency to the selected rule.
- The same list of layout managers like in the Decision Tree Analyser (see Section 4.4.2.3) are available.

CHAPTER 5

EM4J - Performance Evaluation

In this chapter the performance of EM4J is examined. The focus is on the performance of the Advanced Test Case feature. We start with the motivation, a description of the test configuration, the test procedure and terminate with a description of the results.

5.1 Motivation

In Chapter 3 we briefly discussed InGES and its two main procedures, the computation of the initial as well as the dynamic risk factor. EM4J is prototypically designed as a toolbox. The system is not strictly based on the explanation need of InGES however it is exemplarily used to shape the features of EM4J. Additionally InGES is used to perform the evaluation of EM4J. More precisely the Advanced Test Case feature is used to examine the behavior of the initial risk factor computation.
Geologists categorize the characteristics (e.g., slope angle, underground, etc.) of a geological site (in this case we refer to a hillside). This data is the input for the computation of the initial risk factor. We already highlighted the circumstance that disproportional influence of minor characteristic changes is unwanted. This circumstance represents the design motivation of the Advanced Test Case feature. It is based on the ability to automatically test the risk factor computation of such variations and the procedure to generate these variations with this feature. In other words it is designed to answer the question of how stable the result of the initial risk factor calculation with respect to minor changes in the characteristics of hillsides.
During this performance evaluation, we focus on these variations and further back up the need for such a feature. Especially the vast amount of variations that exists make it nearly impossible to manually test the space of all possible risk factors that can be computed to detect errors or inconsistencies in the behavior. Additionally we specifically intend to bring the system to its limits. This test do not reflect an everyday situation! The reason is to assure a certain level of run-time stability of EM4J.
Templates define the structure of the factbase. Consequently they also define the number of possible fact representations. Predefined allowed slot values limit the number of representations.

We will from now on refer to instances since this term is more appropriate. An instance denotes one fact representing e.g., the data for a hillside. We already know from Chapter 3 that the initial risk factor computation needs only one fact. This fact stores the characteristics of the geological side that should be observed with the help of InGES. Therefore, in terms of test cases, one such fact represents one instance respectively one test case. These instances are fed into the Advanced Test Case feature to execute them in order to compute the corresponding initial risk factor.
In the next section we will have a look at the variations respectively instances, their magnitude and where they come from. This is especially important in terms of understanding the need for the Advanced Test Case feature.

5.1.1 Test Configuration

The initial risk factor computation uses one fact to store the characteristics of a hillside. The template underlying these facts defines 38 slots. These slots[1] are as follows (NN indicates that the slot is not set):

1. **vegetation:** NN, meadow pasturage, loose masses, solid rock, forest, single trees
2. **granual material:** NN, block, rock, gravel, sand, silt, clay
3. **subsoil:** NN, malleable, hard
4. **danger recent deformation:** NN, low, medium, high, very high
5. **pieces of rock:** NN, NO, yes recent, yes within 1 year, yes older than 1 year
6. **indicates:** NN, NO, yes recent, yes within 1 year, yes older than 1 year
7. **slope angle:** NN, low, medium, high, very high
8. **danger slope angle:** NN, very low, low, medium, high, very high
9. **slope profile up:** NN, linear, convexly, concavely
10. **slope profile down:** NN, linear, convexly, concavely
11. **slip surface:** NN, NO, YES, 1m, 10m, 100m
12. **material underground:** NN, gravel, clay, basalt, granit, shale
13. **bedding:** NN, prone sliding, intermediate, solid
14. **danger bedding:** NN, low, medium, high, very high
15. **fine grit:** NN, high, very high
16. **danger fine grit:** NN, low, medium, high, very high

[1]The underlined slots are the ones used in the current implementation of the initial risk factor computation.

17. **saturation soil:** NN, low, medium, high, very high, oversaturated
18. **leaning trees:** NN, YES, NO
19. **sickled trees:** NN, YES, NO
20. **leaning rocks:** NN, YES, NO
21. **crack:** NN, yes recent, yes overgrown, NO
22. **rock joint:** NN, not, medium, strong
23. **joint orientation:** NN, dip, strike
24. **insolation:** NN, NO low, medium, high
25. **danger insolation:** NN, low, medium, high
26. **permafrost:** NN, YES, NO, recent past
27. **danger stonechips:** NN, low, medium, high, very high
28. **movement location:** NN, up, down, middle
29. **location of endangered area:** NN, up, down, middle
30. **frost thaw cycle:** NN, YES, NO
31. **depth movement:** NN, shallow, deep
32. **endangered area:** NN, streets, settlements, rivers
33. **date**
34. **time**
35. **place**
36. **risk potential expert:** NN, low_0, lowmedium_1, medium_2, mediumhigh_3, high_4, very_high_5
37. **risk defined:** NN, NO, YES
38. **risk potential:** NN, low_0, lowmedium_1, medium_2, mediumhigh_3, high_4, very_high_5

The template defines for 35 of these slots all allowed values. Each value of this list can be set into the slot. In other words these slots are bound to these values.
In the current implementation of InGES only 28 of these slots are used to store characteristics of the hillside. But only 16 of them are used as an input for the risk factor computation (due to ongoing development). All of these used ones possess predefined values. Therefore a total

of 933,120,000 combinations of slot values exist. This number can be explained through building all possible combinations of the 16 used slots and their allowed values. The information to calculate this number can be retrieved out of the underlined slots above (vegetation(5) x granual material(6) x . . .). In other words with the current setup InGES could observe 933,120,000 different hillsides. Would these values be unbound (without predefined values) the total number would be unlimited since every value can be inserted into each slot.
The remaining seven slots are used to store internal intermediate results during the initial risk factor computation. A summary of the properties of the template used for the initial risk factor computation can be found in Table 5.1.
All of these more than 933 million combinations are of interest. They represent the space of possible input configurations for the initial risk factor computation. However not all of them are valid combinations. A valid combination is one that could actually exist in nature.
Let us have a look at the reason why we show this value. It can be clearly seen that it is impossible to manually test them in order to get a good sense on the correctness of the system. We already made clear that the correctness is vital since the failure costs are extremly high. By failure cost we not only refer to the amount of money to pay for property damage but also the loss of human life. Therefore a tool to evaluate the system's behavior is needed.
InGES must be able to cope with the variations that come with the geologist categorization. Consider the following example in Table 5.2 (The difference between the experts are highlighted).

Total Number Of Permutations	38
Number Of Slots With Predefined Allowed Values	35
Number Of Slots With Without Predefined Allowed Values	3
Number Of Initially Set Slots	28
Number Of Used Slots (Initially)	16
Number Of Used Slots With Predefined Allowed Values (Initially)	16
Number Of Relevant Combinations	$933, 120, 000$

Table 5.1: Properties of the only template in the initial risk factor computation

Two geologists assessed the same hillside. It can be seen that some of the characteristics are categorized differently. Although the system does not know the risk potential assessment of the experts (last row in Table 5.2) it should be in line with them. Disproportional influence on the computed risk factor of such minor changes is unwanted. Therefore these categorization variations have to be investigated. The system has to cope with them.
Investigation of the space of all possible configurations is decisive since inconsistencies and contradictions can be spotted respectively found. To give an example one such inconsistency could be the case if just a minor categorization change (e.g., from medium to high) from one or two characteristics lead to a major change in the computed initial risk factor. Such leaps are especially of interest since they point to likely design flaws in the system.
In the following sections the performance of the Advanced Test Case feature respectively EM4J is highlighted. The goal is to assess the usefulness of the possibility to automatically test the

Slot	**Expert A**	**Expert B**
vegetation	forest	forest
granual material	silt	silt
slope angle	high	very high
slope profile up	linear	linear
slope profile down	linear	concavely
insolation	medium	medium
crack	NO	NO
pieces of rock	yes within 1 year	yes within 1 year
leaning trees	YES	YES
leaning rocks	YES	YES
indicates	yes within 1 year	yes within 1 year
depth movement	shallow	deep
movement location	up	up
material underground	basalt	basalt
saturation soil	medium	medium
slip surface	NO	NO
frost thaw cycle	NO	NO
permafrost	NO	NO
endangered area	settlements	settlements
location of endangered area	down	down
rock joint	not	not
sickled trees	NO	NO
risk potential expert	lowmedium_1	lowmedium_1

Table 5.2: Example of two hillside assessments (non exhaustive)

initial risk factor of various hillsides.
The computer system for the test was an Intel Core 2 Quad Q6600 with 2.4 GHz per core, 3 GB of RAM, equipped with 500 GB hard disk with 7200 RPM and the operating system Windows Vista.

5.1.2 EM4J Implementation

We are now going to show two key features of EM4J respectively the Advanced Test Case feature that have an impact on the time performance. Initially the design was a main memory driven approach. In other words EM4J was designed to store the processed data in the main memory in order to reduce the number of I/O operations. The reason for this main memory driven approach is the fact that in modern computers I/O operations are still one of the most time consuming processes. However to increase the number of processable runs within one performance test some modifications were build in.
The first feature we want to present is the test case archive creation. An archive is generated for each processed test case and are immediately written to the file system instead of keeping them

in the main memory. The reason for this design is that these archives are very space comsuming and must be written to the file system.
The second feature we want to present is the report generation. A report can be generated for each test run or one for the whole test case. If one report for the whole test case is generated the data is kept in main memory. For huge numbers of test instances within one test case the option to create one report for each run has to be chosen. The reason for this circumstance is the fact that for example a single PDF report for 165,888 test instances alone consumes approximately one GB of hard disk space.

No. of Instances	Memory Used	Total HD Space[1]	Time	Result
27,648	400MB	11.0GB	0d 02h 30min	ok
27,648	400MB	11.0GB	-	nok[2]
55,296	400MB	22.1GB	0d 06h 45min	ok
55,296	400MB	22.1GB	-	nok[2]
110,592	490MB	44.8GB	0d 18h 0min	ok
110,592	490MB	44.8GB	-	nok[2]
165,888	600MB	66.0GB	2d 01h 0min	ok
165,888	600MB	66.0GB	-	nok[2]
248,832	-	-	-	ok[3]
497,664	-	-	-	ok[3]
995,328	-	-	-	ok[3]
1,190,656	-	-	-	ok[3]
3,317,760	-	-	-	ok[3]
7,776,000	-	-	-	nok[4]

[1] Hard disk space including archives and reports consumed.
[2] EM4J (creating one report for all test instances) stopped due do missing free main memory to work with.
[3] EM4J completed the test case creation. The test scenario was not executed due to hardware contraints.
[4] EM4J stopped creating the test case scenario due to missing free main memory to work with.

Table 5.3: Test runs performed during the performance test

5.2 Results

The first topic we want to highlight is the option to generate one report for the whole test case. Using this option the number of instances was reduced to approximately 27,000 contained in one performance test and the system still stopped to work. This was due to the fact that no more free main memory was available to work with since all the generated report data was stored in it. This number is not very good considering the fact that by the template definition of the initial risk factor computation 933,120,000 combinations respectively instances are possible. In other

words with the 27,000 instances the user would need approximately 34,000 tests to investigate all possible combinations. On the other hand using EM4J with the option to create a report for each instance the testcase containing 27,000 instances is successful. Therfore we increased the workload.
Detailed information on the tests performed is listed in Table 5.3. In this table we can be seen that the system is stable. The only bootleneck available is the design decision that all generated test instances are stored in main memory and not created during the execution of the test case. The advantage for the user is a GUI that enables him/her to investigate the setup of the test case. The user is able to check the created facts and their assignment to jobs and runs. However this restriction is desired. The upper limit on the number of facts contained in one test is around three million. We did not perform such a huge test since the amount of hard disk space needed is not feasible for most standard modern computer (ca. 1,4TB). But in principle EM4J is capable to perform such huge tests if the right hardware is in place. Even more instances are possible if the right hardware is in place.
Now let us have a look at the actual performance of EM4J. There are some points that immediately attract attention:

- **In Memory Jobs:** This is the only real design constraint prevalent in EM4J. The idea behind this was that the user should have the ability to examine all his created instances respectively combinations whether the setup is correct. This restricts the total amount of instances that can be created for a specific test case. But since RAM for computers is not that expensive anymore the impact is mitigated because the user has the ability to upgrade the system (if absolutely needed). Otherwise a massive test case can be simply splitted into a few workable portions.

- **Report Creation:** The option to create a report for every instance eliminated a memory problem since they are immediately written to the file system. However if the user wants to create a single report for a specifc test the number of instances is restricted again. This is due to the fact that the results collected and consequently written into the report are stored in memory. The actual amount of instances operable is dependent on the KBS. The more rules fired respectively facts modified during a KBS execution the smaller the number of instances that can be contained in the test.

- **HD Space:** EM4J needs a lot of hard disk space. The actual amount needed again depends on the KBS (number of rules/facts, number of firings, etc.). But since hard disks are quite affordable nowadays the impact of this issue is kept within limits. If EM4J runs on a restricted system the option to calibrate the length of the archive history gives the user the capability to manage the limited amount of space. Consequently the risk of running out of hard disk space is mitigated.

- **CPU Usage:** EM4J takes full advantage of the power provided by the CPU. However, multi-cores are not supported! The utilization of multi-cores would not improve the performance of EM4J in its current design. The reason is that the system has to process huge amounts of data. Consequently the I/O interface and not the CPU is the performance bottleneck. Only an upgrade in EM4J's data management (e.g., parallel processing, incremental processing, etc.) would enable the utilization of multi-cores. However the improvement is still questionable since synchronization methods have to be implemented in order to avoid data loss. Therefore the question arises whether the performance gain is worth the effort.

CHAPTER 6

Conclusion

The motivation for EM4J is to support the users of InGES in assessing the properness of the system's computed risk factor. The plain rule trace is hard to examine in order to check the correctness of the computed risk factor. EM4J provides an abstracted, filtered and graphical enriched decision tree to relieve the user from digging into hundreds of sheets of paper (recall the trace example out of Chapter 1). This tree mirrors the reasoning performed by the system.
Beside the tree, testing the overall correctness of InGES represents the second core feature of EM4J. This testing capability is facilitated with the Advanced Test Case feature. It enables the user to automatically test all possible input configurations of the system to find inconsistencies respectively contradictions.
In Chapter 2 an epitome about the problems that surround the generation of explanations is given. Generally speaking the goal is simple. The user has to be fed with apt information tailored to his needs. Different views tackling theoretical definitions and principles which led to various approaches are listed. Wick's approach matched to our problem at hand. It defines the characteristics of how to generate explanations in a way suitable for our purpose.
Additionally the shape of our explanations is influenced by three major factors:

1. **User Group:** We are confronted with a user group composed of highly skilled technicians namely domain experts as well as knowledge engineers. The domain experts are mainly geologists who possess high expertise about the environment. InGES is designed to support them in their domain. The knowledge engineers are a mixture composed of software developers and geologists as well. Therefore a high percentage of domain knowledge is brought along with our user group. This circumstance mitigated the need to explain the domain with EM4J. The user already understands the problems respectively issues of as well as the domain itself. To give an example, there is no need to explain the questionnaire filled out by the geologists. EM4J can solely focus on why and how InGES computes this or that risk factor for a geological site. In detail, which rules are fired, which facts lead to the firing of the rule and how they influenced the computed recommendation.

2. **Effects of Expertise:** Section 2.2.2 showed a study about the effects that the user's expertise has on desired explanations. Justifications, strategic explanations as well as domain explanations are the preferred types desired by novice users. In contrast, expert users favor how-type explanations to conceive the reasoning process. Reasoning traces are a highly liked approach to fulfill this need. The upgrade to plane reasoning traces are decision trees.
   This study influenced the design of EM4J to explain how and why InGES computes this or that risk factor for a geological site.

3. **InGES's Application Domain:** InGES works in a high failure cost environment. Therefore the correctness of the computed risk factor is absolutely inevitable. Only then the user gains confidence in the system.
   This circumstance lead to the design of the Advanced Test Case feature. It gives the user the capability to automatically test the course of behavior of InGES in order to investigate if the already mentioned disproportional influences exist.
   The reason to implement the Advanced Test Case feature is simply the fact that high failure costs are prevalent in InGES's application domain. High failure costs describe the costs that occur in case of a false classification by InGES. The user is aware of this circumstance and very sensitive to everything other than his own perception since there is a lot at stake (e.g., human live). Conclusively InGES is useless if geologists do not trust its computed risk factor.

In principal the approach of Wick et al. was chosen to design EM4J. This approach describes three major factors that form the shape of explanations. A tight coupling of the inference procedure with the explanation facility in connection with a process and verification-orientation for highly skilled users is proposed by this approach. Geologists constitute the user group of InGES and EM4J consequently we are dealing with exactly such highly skilled users. EM4J is highly coupled with the inference engine (for more information see Chapter 3) fulfilling this aspect of the proposal of Wick et al. The process-orientation is fulfilled with the ability to investigate decision trees. EM4J allows the user to investigate why and how InGES computes this or that risk factor for a geological site. In detail, which rules are fired, which facts lead to the firing of the rule and how they influenced the computed recommendation. Additionally the verification-orientation is supported through this feature too. The user gains knowledge why and how the risk factor is computed and therefore is able to assess its correctness.
In summary, based on the user group the effects that their expertise have and the application domain of InGES EM4J is designed as a highly technical transparent process oriented explanation module toolbox. The user is supported with a set of tools to investigate the decision tree, the rule dependencies and the capability to automatically test InGES with differing input. These tools should accomplish a state of satisfaction for these highly skilled experts using InGES.
Last we want to emphasize that EM4J exploits the underlying hardware very well. With the performance evaluation of the Advanced Test Case feature this fact is further underlined. The Advanced Test Case feature processes huge amounts of data and performs at the limit of the hardware.

6.1 Future Work

We now briefly want to discuss future enhancement topics of EM4J. Further investigation can be performed on:

- **Multi-core CPU Architecture:** The performance of the decision tree generation as well as the Advanced Test Case is at this point not utilizing the multi-core architectures of modern computers. The feasibility of multi-cores is something to investigate on in future.
- **Visualization:** The GUI presentation of huge graphs is certainly a problem. The representation can be confusing for the user especially if there are lots of nodes and vertices in the graph. Investigation on clustering methods can be performed in order to mitigate this problem. Especially collapse and shrinking methods similar to the idea used during the calculation of strongly connected components[1] in graph theory can be of interest.
- **Rule Dependencies:** In future versions the Rule Dependency Analyser can be subject to enhancement. In the current implementation only a static dependency evaluation is performed (For more information we refer to Section 3.4). Variables, functions, user input, etc. are not included in this evaluation. Therefore the feature that allows the user to define the value of variables and consequently evaluate the dependencies with them can be added. This feature would enhance the rule dependency analysis significantly. However the implementation effort for such a feature is high and has to be considered.

[1]http://www.cs.berkeley.edu/~vazirani/s99cs170/notes/lec12.pdf

Bibliography

[1] *GeoMoS*. http://www.leica-geosystems.com.

[2] *GOCA*. http://www.goca.info.

[3] *OASYS*. http://www.vce.at/oasys/.

[4] B. Chandrasekaran, M. C. Tanner, and J. R. Josephson. Expert systems: the user interface. chapter Explanation: the role of control strategies and deep models, pages 219–247. Ablex Publishing Corp., Norwood, NJ, USA, 1987.

[5] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Trans. Graph.*, 15(4):301–331, 1996.

[6] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[7] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991.

[8] K. Fujisawa. Monitoring technique for rock fall monitoring. Technical report, 2000.

[9] S. D. Gregor. *Explanations from knowledge-based systems for human learning and problem solving*. PhD thesis, University of Queensland, Brisbane, 1996.

[10] S. D. Gregor and I. Benbasat. Explanations from intelligent systems: theoretical foundations and implications for practice. *MIS Q.*, 23(4):497–530, 1999.

[11] S. Hughes. Question classification in rule-based systems. In *6. Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems*, pages 123–131, December 1986.

[12] W. Lehnert. A conceptual theory of question answering. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 1*, pages 158–164, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[13] A. Majchrzak and L. Gasser. On using artificial intelligence to integrate the design of organizational and process change in us manufacturing. *AI & Society*, 5:321 – 338, October 1991.

[14] J. Mao and I. Benbasat. The use of explanations in knowledge-based systems: Cognitive perspectives and a process-tracing analysis. *J. Manage. Inf. Syst.*, 17:153–179, August 2000.

[15] C. J. Martincic. Que: an expert system explanation facility that answers "why notttypes of questions. *J. Comput. Small Coll.*, 19:336–348, October 2003.

[16] B. Meyer. Self-organizing graphs - a neural network perspective of graph layout. In *GD '98: Proceedings of the 6th International Symposium on Graph Drawing*, pages 246–262, London, UK, 1998. Springer-Verlag.

[17] S. Moen. Drawing dynamic trees. *IEEE Softw.*, 7(4):21–28, 1990.

[18] J. W. Payne, J. R. Bettman, and E. J. Johnson. *The Adaptive Decision Maker*. Cambridge University Press, 1993.

[19] M. Rabinowitz, R. Glaser, and National Institute of Education (U.S.). *Cognitive structure and process in highly competent performance [microform] / Mitchell Rabinowitz and Robert Glaser*. Learning Research and Development Center, University of Pittsburgh, Pittsburgh, 1986.

[20] A. Reiterer, M. Lehmann, M. Miljanovic, H. Ali, G. Paar, U. Egly, T. Eiter, and H. Kahmen. A 3d optical deformation measurement system supported by knowledge-based and learning techniques. *Journal of Applied Geodesy*, 3:1–13, 2009.

[21] D. Richards. Knowledge-based system explanation: the ripple-down rules alternative. *Knowl. Inf. Syst.*, 5:2–25, March 2003.

[22] M. Scaioni, A. Giussani, F. Roncoroni, M. Sgrenzaroli, and G. Vassena. Monitoring of geological sites by laser scanning techniques. In *IAPRSSIS*, volume 35, pages 708–713, 2004.

[23] F. Sørmo, J. Cassens, and A. Aamodt. Explanation in case-based reasoning—perspectives and goals. *Artif. Intell. Rev.*, 24(2):109–143, 2005.

[24] W. R. Swartout and J. D. Moore. *Explanation in second generation expert systems*, pages 543–585. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1993.

[25] T. Vicovac, A. Reiterer, U. Egly, T. Eiter, and D. Rieke-Zapp. Knowledge-based geo-risk assessment for an intelligent measurement system. In Max Bramer, editor, *Artificial Intelligence in Theory and Practice III*, volume 331 of *IFIP Advances in Information and Communication Technology*, pages 215–224. Springer Boston, 2010.

[26] M. R. Wick and W. B. Thompson. Reconstructive expert system explanation. *Artif. Intell.*, 54(1-2):33–70, 1992.