Expert System Framework for Supply Chain Optimisation

by
Daniel O'Grady

Masterthesis
18. April 2016

Area of Studies	Computer Science
Matriculation Number	3787359
Principal Supervisor	Prof. Dr. Torsten Grust
Co-Supervisor	Dr. Britta Dorn

ABSTRACT

Using an *enterprise resource planning* tool (ERP) is quite common these days in many companies, regardless of their size. While a well-maintained ERP has many benefits for the company, incorrect or just careless use carries the risk of actually worsening the stockkeeping. Discovering such flaws and optimising the storage requires expert knowledge and careful evaluation and analyses. This thesis aims to develop a modular framework for tools to automate the evaluation of ERPs, or at least support human analysts in this task. To do so, the framework provides a standardised workflow for reading data from arbitrary ERPs, transforming it into a common format and running analyses on them. In addition to visual analyses, a rule-based expert system is introduced, allowing for relatively easy development of an ERP-specific ruleset.

DECLARATION

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbstständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

19th April 2016, Reutlingen,

CONTENTS

LIST OF FIGURES

ACRONYMS

AI Artificial Intelligence

ES Expert System

ERP Enterprise Resource Planning

FARAD Framework for Arbitrary Ressourcesystem Analysis and Data visualisation

DynaMO Dynamic Material Optimisation

DSL Domain-specific Language

1

INTRODUCTION AND THEORY

1.1 PURPOSE

For many medium-sized to large companies, it is custom to employ *enterprise resource planning (ERP) tools* to improve stockkeeping and organise the supply chain[1]. While a well-kept ERP can help optimise recurring tasks, misapplying it can actually have negative effects. Consider an inexpensive spare part that is ordered on a regular basis for repair shop to provide fast access upon maintenance. The booker therefore buys them ahead every month to avoid shortages. As it turns out, another booker did the same and now the spare parts are accumulating. This may be detected in the annual stocktaking, but chances are that many superfluous units have piled at that point. At worst, the parts have some sort of expiration date, making it impossible to plan with foresight.
Another frequent problem is incomplete or even missing data in an ERP. Some entries for storage movement or trunk data might be missing, due to carelessness of the user or simply because the data is not available (yet). Circumstances like these require consultants, who dig through the storage and discover weak points and provide customised suggestions for the company.

This thesis aims to develop a modular, computer-aided solution in the form of a framework (FARAD) to support companies in improving their supply chain and eliminating flaws in their storage management. A fully functional prototype (DynaMO) is being produced alongside, involving all the abstraction layers provided by the framework.
The original intent was to create a single *expert system* (ES) to provide automated suggestions to improve the stockkeeping and resource organisation. As such suggestions require further work, such as data reading, sanitisation and normalisation, the project grew into a whole framework. But the main goal of creating an ES still prevails. This document is therefore structured into an introduction part, illustrating expert systems as a whole, followed by the actual implementation of FARAD, including the structure and additional analysis methods it provides.

1 The system containing all participants and processes included in supplying a customer with a product or service.

Agriculture Law	Business Manufacturing
Chemistry	Mathematics
Communications	Medicine
Computer systems	Meteorology
Education	Military
Electronics	Mining
Engineering	Power systems
Environment	Science
Geology	Space technology
Image processing	Transportation
Information management	

Figure 1: Categories of ES's as identified by Durkin in [3, p. 14]

1.2 EXPERT SYSTEMS

1.2.1 *A Brief History*

With the rapid progress of computing capabilities throughout the early to mid 20th century, computer-aided calculation and evaluation gained popularity in the scientific and military fields. [3, p. 6-9] It also spawned the fascination with intelligent computer programs being able to emulate human reasoning, and effectively solving general problems. This movement of *artificial intelligence* would experience a brief period of vast attention [3, p. 8, 9] followed by it's downfall as it became apparent that AI was not applicable to real-world problems at that time.[3, p. 8, 9]
The idea of intelligent computers regained momentum when NASA approached researchers at Stanley University to develop a system for an unmanned space probe, that would analyse the molecular structure of Martian soil, which would contain molecules of high diversity. The developed system, DENDRAL, tried to emulate the behaviour of an expert chemist, approaching the task with heuristics to narrow down the possibilities and apply expert knowledge to solve the problem. [3, p. 11]
From that point on, ES's gradually found their way into a multitude of fields, which were categorised by Durkin as displayed in Figure 1.

1.2.2 *Components and Structure of an Expert System*

1.2.2.1 *Structure*

Developing and using an expert system typically involves multiple parties: the developer, who implements the system, the end user, who uses the system to solve some task, and the expert, who provides the rules and heuristics he would apply if he had to tackle the problem himself.

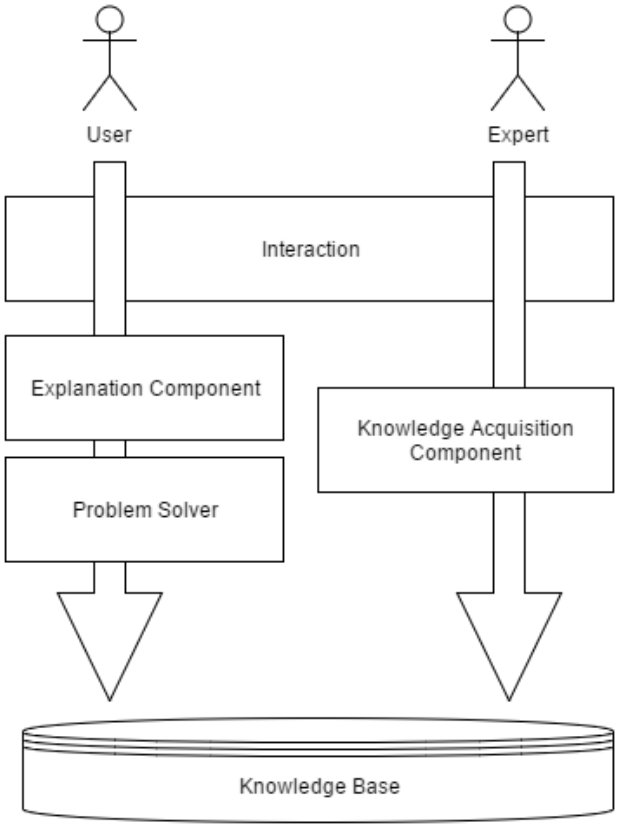

Figure 2: Architecture of an ES, showing the expert's and the user's view on the knowledge base.

The developer himself is not necessarily an expert in the field the ES operates in, so usually, he has to consult experts. Those might also not be familiar with software development, so it is reasonable to provide two non-developer-views of the system: the *user's view* and the *expert's view*. [2, p. 24, 1, p. 7]
While the expert's view provides means to modify the underlying knowledge base from which the solutions are generated, the user's view enables the user to query the system for support on a question in the ES's field. It might also contain an explanation component, that translates the solution into a language that is easily understandable for the user. Both views are depicted in Figure 2.

1.2.2.2 *Knowledge Representation*

Because knowledge is a rather abstract concept, we need some way to represent this information that reflects the knowledge itself and makes it easy to apply to problems. Cremers et al. present various ways to do so in [1, chapter 2.3]. They also evaluate those representations with respect to their usability in certain situations in [1, table 2.1], asserting that for incomplete, inconsistent knowledge, for small-

```
IF:
      identity of the organism is unknown AND
      stain is gramneg AND
      morphology is rod AND
      the aerobicity  aerobic
THEN:
      type of the organism is enterobacteriaceae
      with certainty (.8)
```

Figure 3: Rule 037 of MYCIN, based on the depiction by Karst. [2, p. 36]

to medium-scaled systems, that should be easy to modify and extend, a rule-based approach is a suitable method.
This meets our requirements as we have to deal with probably incomplete data where special rules have to be provided for each company, or at least company type.

RULE-BASED SYSTEMS One very early representation of knowledge, predominant in the 1960s and 1970s is the *rule*-based representation. Rule-based systems strive to emulate a human expert, that applies some rules of thumb as Durkin explains. [4, p. 24]
He identifies three components in a rule-based system:

1. *Knowledge*, in form of rules, which captures the heuristics of experts in the domain the ES operates in.
2. *Working memory*, containing the relevant facts and conclusions (the current problem).
3. The *inference engine*, which combines knowledge and working memory to derive conclusions.

Rule-based systems also come in two flavours: backwards- and forward-chaining. While the former starts with a hypothesis and works it's way to a solution and is therefore used for diagnostic problems, forward-chaining iteratively derives additional information from the provided facts. It is used in planning situations, where there might be very little initial knowledge, that is gradually enriched through the ES itself. An example for a rule as used in the early ES MYCIN is shown in Figure 3. This approach is frequently used in *blackboard architectures*. In a blackboard scenario, a group of experts is confronted with a problem, with only a blackboard to communicate with each other. While not every expert might be able to immediately contribute to the solution, another participant could provide a piece of information, enabling the first expert to contribute after all.

2

FRAMEWORK AND TOOL

2.1 FARAD AND DYNAMO

FARAD is a framework providing a general workflow to analyse ERPs by defining steps in which the data from a source is processed, stored and then analysed. A core mechanic of FARAD is a built-in expert system that uses a rule-based forward-chaining approach, combined with a blackboard architecture to enrich the data collected from an ERP. The data will be categorised, using well-known analysis methods from the field of logistics. All information gathered and constructed by FARAD can be visualised, allowing experienced human consultants to provide suggestions and actions that should be taken. Those suggestion patterns can be formalised again to be expressed in the form of rules as described in Paragraph 1.2.2.2. Those rules can then be used to automatically generate suggestions to support the consultant or completely replace them.

DynaMO is the prototypical tool that is being developed in the scope of this thesis and is built on top of the framework, demonstrating it's capabilities in an exemplary manner. It implements the architecture specified by FARAD as a proof of concept and is described in detail in the following sections. Much like a dynamo, the tool tries to generate light to shed on an ERP - hence the name.

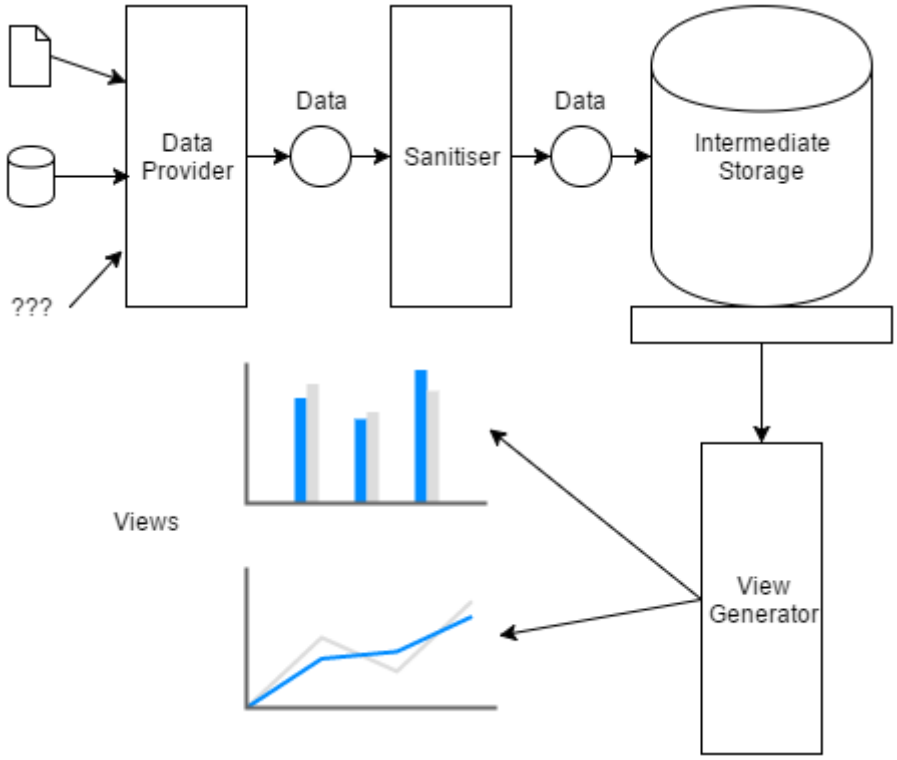

Figure 4: General architecture of the framework. Illustrated is the flow of data from arbitrary data sources through a sanitiser to the intermediate storage, from where it is used to generate views.

2.2 ARCHITECTURE OF FARAD AND DYNAMO

The software accompanying this thesis is developed as a framework for ERP evaluation. As such, it is divided into several independent components, each serving a very specific purpose to allow for dynamic substitution of any component while preserving the general workflow for the evaluation. DynaMO itself is therefore merely an instance built on top of the framework. The components of the framework are illustrated in Figure 4 and will be explained in the following chapter, each being illustrated through the implementation of DynaMO.

2.2.1 *Data Provision*

A vital feature of FARAD is the compatibility with arbitrary ERP systems. FARAD therefore provides an abstraction layer between the actual data storage of the tool and the evaluated system.
While the data can be provided in any form, it should contain certain information, grouped into data sets, in order for the program to dis-

play the basic views correctly. To avoid constraining the framework to a limited list of ERPs, a minimal set of mandatory information has been identified to provide all basic functionality that comes with the framework:

- **data trunk:** containing all basic information about an article. Marked by the `ITrunk`-interface.
  - **material number**: a unique identifier for an item.
  - **description**: a description string for a human user.
  - **safety stock**: buffer stock of an item that has to be kept at any time.
- **movement data:** information about transfers of the item. FARAD only assumes in- and out-transfers. Other movements, such as moving an item within the factory to another storage place, has to be mapped onto a "neutral" movement. Movements have to implement the `IMovement` interface.
  - **date**: the date when the transaction took place.
  - **direction**: direction in which the item was moved. Either in (acquisition), out (consumption) or neutral (special movements, that are not captured by the framework).
  - **quantity**: how much of the item was moved in this transaction.
- **stock information:** information about the current stock, marked by implementing the `IStock` interface.
  - **quantity**: the current, total quantity, covering all sorts of stocks. [1]
  - **value**: the total value of the current stock.
- **delivery information:** information about the delivery of an item which is denoted by implementing the `IDelivery` interface.
  - **supplier**: information about the supplier that delivers the article.
  - **delivery time**: time in running days it takes the supplier to deliver the article.
  - **price**: how much the supplier charges for one unit of the article.

Any implementation can specify and require additional data sets and extend the existing sets for additional views. Within the scope of this

1 Some ERPs differentiate between "available" stock and stock, that is present, but currently not usable.

```
--
|FINDEX|BEST| MATNR| WERK| DISPO| SAFEB| DESC|
--
|0|23.0000| 2| 3| 5| 100.0000| 8-fach Darlington Arrays|
|0|500.0000| 1| 2| 1| 100.0000| Kohleschicht-Widerstand|
|0|2500.0000| 5| 3| 4| 3000.0000| 4 EXCL.NOR|
|0|1.0000| 5| 1| 1| 200.0000| Zener-Diode|
```

```
--
|FINDEX|MATNR| WERK| DISPO| MENGE| BWART| DATE|
--
|0|2| 3| 5| 100.0000| 261| 31.12.2015|
|0|2| 3| 5| 34.0000| 261| 25.01.2016|
|0|5| 3| 4| 1230.0000| 0| 19.11.2015|
|0|5| 1| 1| 2500.0000| 103| 30.01.2016|
|0|5| 3| 4| 3400.0000| 261| 10.11.2015|
```

Figure 5: Randomly generated data that can be used for DynaMO mimicking a plaintext export from an ERP. The data is divided into two separate files. The first file holds basic information, such as the safety stock and the description. The second file contains the movement information for each part. The two files are joined on their common index fields `MATNR`, `WERK` and `DISPO` to build coherent records that are stored in the intermediate storage as shown in Figure 6.

thesis, a provider for an unmodified SAP ERP has been implemented, utilising an export of selected tables in plaintext format as illustrated in Figure 5. The provision step is designed with sanitisation in mind: some ERPs may require us to check the loaded data for validity before actually using them to run analyses on them. This has to be worked out for each ERP individually and is therefore part of Section 3.1.

2.2.2 *Intermediate Storage*

The intermediate storage is responsible for storing the data in a consistent format after the provider has read all records for the analysis. From an architectural point of view the tool is compatible with any such storage, ranging from a plaintext storage (possibly in XML or JSON), in-memory databases like SQLite or fully-fledged database systems, such as PostgreSQL or MySQL. The only requirement is an access class, denoted by implementing the `IIntermediateStorage` class.

To keep DynaMO as lightweight as possible, without requiring further installation of additional software, such as an SQL-server, the choice fell on the object database Perst, enlarging the tool's binaries by only a few (< 10) megabytes.

As opposed to classic relational databases, object databases allow for

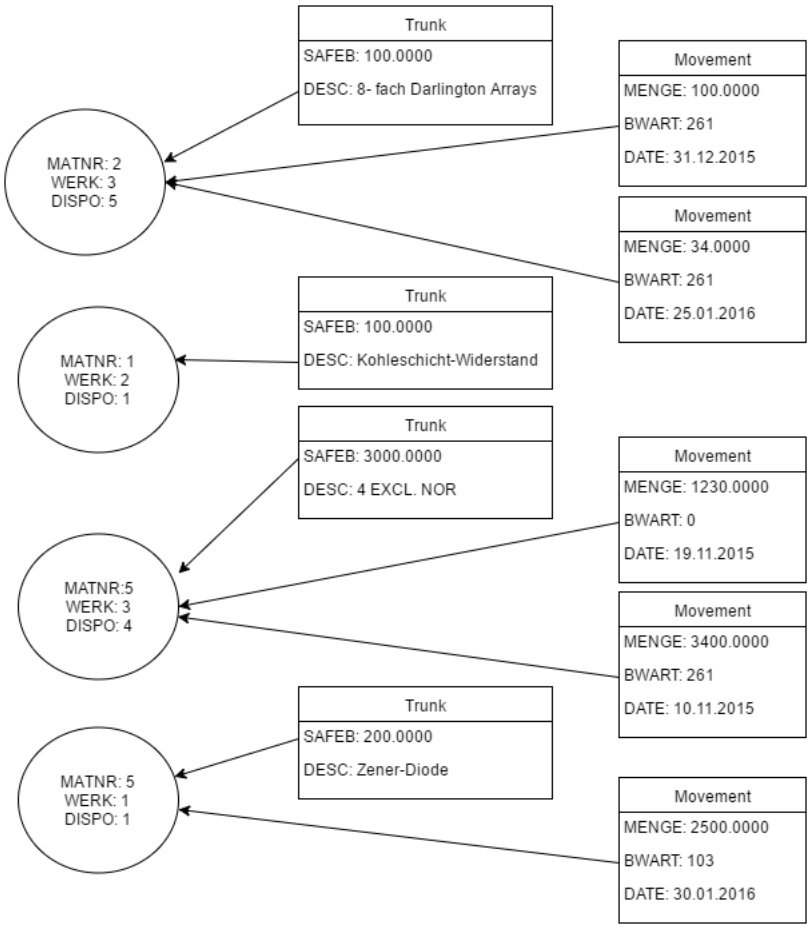

Figure 6: Object model of the sample data displayed in Figure 5 after importing from plain text.

relatively easy retrieval of complex objects consisting of many smaller objects, while still offering features like queries over arbitrary fields, indices and, in the case of PERST, persistent storage. It also spares us the trouble of defining the data schema for each ERP as some fields might differ in length or type, or records could carry more information for additional analyses, depending on the source system.
To organise data entries from our source system, each bundle of information as specified in Section 2.2.1, is stored in an own object. All objects, sharing the same material number are aggregated into a data record. An index over the material number allows for fast retrieval of all data relevant for a specific material.

2.2.3 *Viewgenerator*

To keep the view on the intermediate storage extendible, the framework provides an abstraction layer: the viewgenerator. Each view must implement the `IViewProvider` interface to generate a visual presentation in a `JComponent`. The basic views utilise the analysis described in Section 2.4 and visualise their results.

2.3 DECISION-MAKING

FARAD provides a suggestion module, that evaluates items based on their state at the time of the snapshot and suggests a desirable target-state. To enable modular behaviour, that can easily be extended a rule-based approach is implemented within the framework. A rule is basically a conditional action as described by Cremers et al. [1, p. 16-20]
When working with rules, the order in which they are applied may be of importance. Consider two rules R_1 and R_2, where R_2's action will prevent R_1 from triggering. This produces a classic race-condition as depending on which rule is executed first, we would get different results.

The first considered solution to this was making use of the query-planning algorithm employed in PostgreSQL (see [11]). It deals with the challenge of performing an n-ary join on database tables in a fast-paced manner, considering different join-operations and -orders. To achieve this, the joins are iteratively computed, evaluated and reused through dynamic programming.
This approach turned out to be a dead end as it became apparent that rules, other than the tables in a join, can be applied multiple times, basically defeating the purpose of dynamic programming where the search space would decrease with the number of unjoined tables.
It was therefore discarded in favour of a moderated, rule-based decision-making mechanism. This module consists of two components: the *rules*, which are the *knowledge* as described in Section 1.2.2.1 and the *moderator*, which takes on the role of the *inference engine*. They then create a decision tree, which, together with a context object is the *working memory* in this case.

RULES Whenever an item gets passed to this module, each rule will have a look at it and the context, deciding whether it is interested in it or not. Rules express their interest I in the item i and the context c by yielding $I(i, c) \in [0, 1]$, which will be used by the moderator to determine whether the rule will be applied. Interest can either be returned in a static fashion or can be calculated dynamically by looking at interesting aspects of the passed item.
Once the rule receives permission to apply itself to the item, it can either modify the context (the blackboard), or create a new revision of the item, transform the copy in some way and then append it to the revision tree of the item. This can be applied recursively with each rule and each revision to generate a tree whose leaves are revisions of the original item, which (hopefully) can be considered an improvement. The tree then holds a maximum of $|Rules|^{d-1}$ nodes

with d being the depth of the tree. Alongside with each revision, the rule which created it is stored in the node of the tree. This enables a human user to evaluate the path the system took or initiate actions to actually get from the current state of the item to one of the leaf-revisions.

Writing rules is also the interface for experts to provide the tool with their knowledge: each rule is created by writing a Java-class and providing the evaluator with a path to the compiled classes, allowing for dynamic rule-loading. As this requires the expert to know at least the fundamentals of the programming language the tool was written in, this leaves room for improvement as described in Section 3.1.

MODERATOR As the aforementioned procedure can potentially create an endless cycle where two or more rules cancel each other out or where the tree just reaches unwieldy dimensions, a moderator is required to control the growth of the tree and stop branches which don't look promising.
To ensure that the algorithm terminates, the moderator maintains a threshold $\in [0,1]$ that grows with each level of the revision tree as illustrated in Figure 7. A rule is only allowed to apply itself if it's interest passes the threshold. The growth of the threshold is
$max(c, 1 - \langle \text{ interest of all rules on the last level} \rangle)$,
where c is a constant $\in]0,1]$.
Low overall interest therefore leads to quick termination of the algorithm, while even with an average interest of 1, the moderator will only allow for a tree depth of $d = \lceil \frac{1}{c} \rceil$.

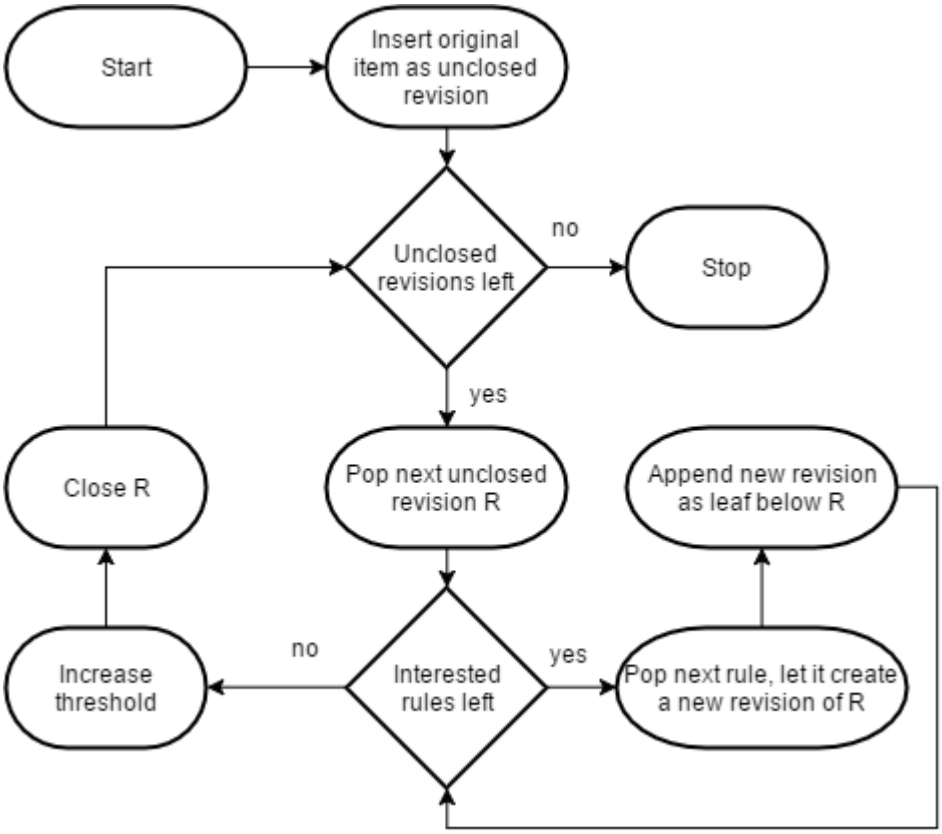

Figure 7: Flowchart of the decision-making algorithm. It illustrates how the revision-tree for one input item is being generated.

```
class ReachView<
    T extends ITrunk,
    S extends IStock,
    M extends IMovement,
    D extends IData
        & ITrunkMarker<T>
        & IMovementMarker<M>
        & IStockMarker<S>
> { ... }

public interface IMovementMarker<M> extends IDataMarker {
    List<M> getMovements();
}

public interface IMovement {
    public enum MovementDirection {
        IN, OUT, NEUTRAL
    }

    Date getDate();
    double getAmount();
    public MovementDirection getDirection();
}
```

Figure 8: The implementation of the reach analysis requires records to provide trunk information, movement data and the current stock. The analysis itself, therefore forces the Java class that reflects the record to implement the appropriate interfaces, in order to be passed as parameter for the analysis. As an example: the reach analysis requires its data to implement (amongst others) `IMovementMarker`, which in turn offers a method that yields a list of `IMovements`, which encapsulates all information that is relevant for one movement.

2.4 ANALYSIS METHODS

FARAD provides the user with several views on the imported data. Each view can either be visualised or used as a foundation for further analysis. As such, the ABC/ XYZ analysis can either be displayed for visual analysis or be utilised by the expert system to create a suggestion analysis. As FARAD does not specify a definitive data schema, each analysis has to specify what information they need from a record to work properly. This is achieved by requiring the data to implement certain interfaces through which specific attributes can be acquired. When expanding the functionality of FARAD with an additional analysis, which could require data that is not included in the current definition of a data record, new interfaces can be created to add the missing attributes without breaking the existing analyses. The views and their underlying analysis are explained in the following sections.

Class	Base price	Importance	Access
(A)	expensive	important	often
(B)	medium expenses	less important	sometimes
(C)	cheap	unimportant	rarely

Figure 9: Examplary classifications for an ABC analysis.

2.4.1 *ABC Analysis*

The *ABC analysis* is a method to classify a set of objects into (usually three) classes, depending on some quantifiable criterion. [9, p. 33, 10, p. 12] Example classifications are shown in Figure 9.
Categorising data this way facilitates the analysis of underlying problems as the analyst can focus on the more important categories to identify critical objects. Vollmuth also suggests that, based on the observations from the ABC analysis, future orders, sales and storage place usage can be improved. [8, p. 21] The classes, of course, have to be well-defined through thresholds indicating when an item falls into which class. Huber and Laverentz observed that in most cases, the distribution among the three classes follows the Pareto principle [2]. They report that, for example, A-items cover about 80% of the cumulated value, while the remaining 20% are covered by items from the B- and C-category and propose an 80/15/5-division when deriving a set of classes. [9, p. 35] In the case of DynaMO, the classes are configurable to match the user's needs.

2 Also known as the 80-20-principle, which means in this case that 80% of the revenue is generated by 20% of the articles. See [5, p. 106, 107].

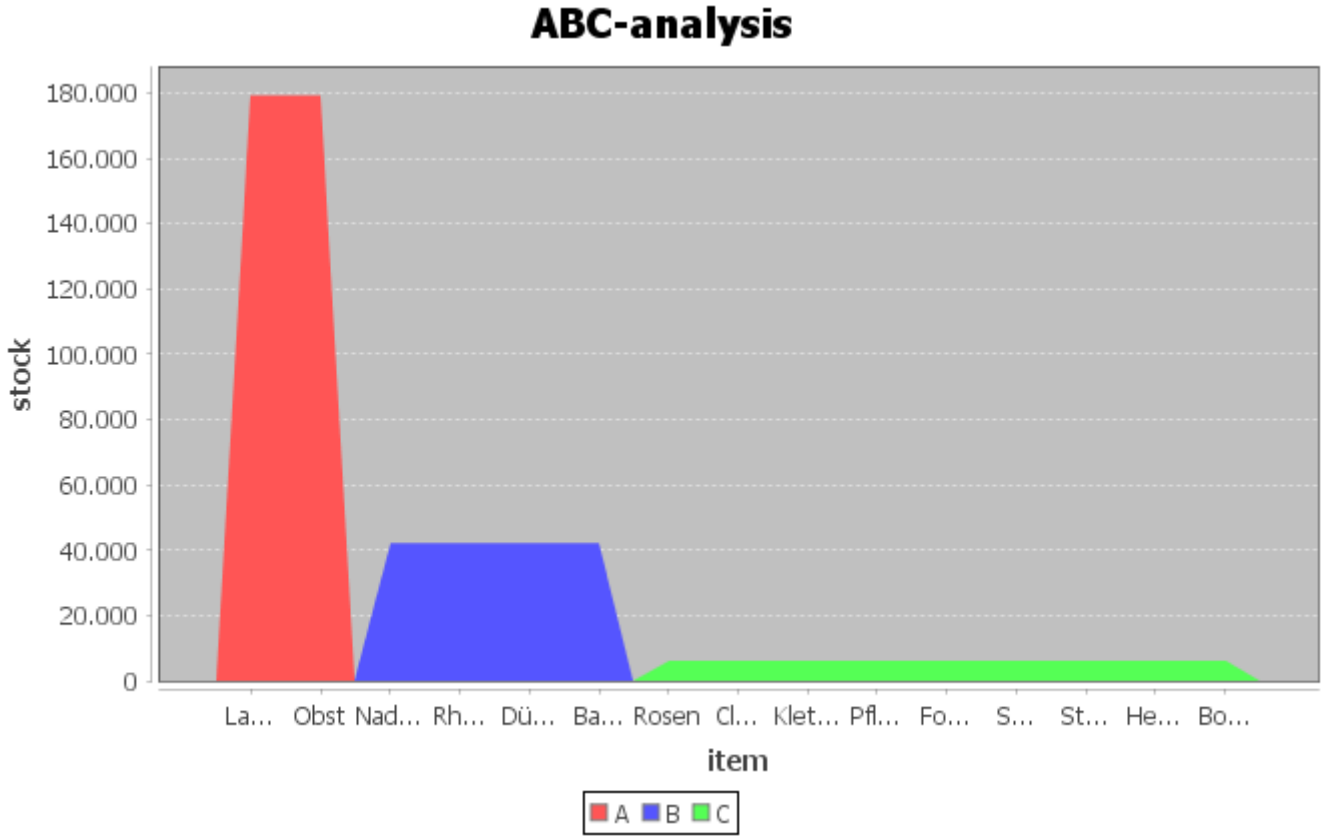

Figure 10: Screenshot from DynaMO, showing an ABC analysis with sample data. The y-axis shows the cumulated stock of each category, demonstrating how two articles make up the better part of the total stock. The partitioning here is A=0.65, B=0.2, C=0.15.

Class	Usage	Example	Stocking
(X)	predictable	screws, soap, oil	order just in time
(Y)	medium predictability	paint, spare parts	keep in stock
(Z)	bad predictability	engines	order on demand

Figure 11: Examples for X-,Y- and Z-categories for a car repair shop.

2.4.2 *XYZ Analysis*

The *XYZ analysis* visualises the predictability of the demand of the analysed objects. While objects in the X-category have a stable predictability, making it easy to foresee the usage and plan the sourcing accordingly, the demand of objects in the Y-category tends to vary to some degree but is still rather stable. Z-objects are even more unpredictable as they are sparsely used, but still needed from time to time. Considering the results of this analysis enables the user to order certain objects "just in time" (X-objects), while other objects are kept in storage (Y- or Z-objects) or are even ordered on demand (Z-objects). [9, p. 36] A sample classification can be seen in Figure 11.
To express the "predictability" of an item, we calculate the coefficient of variation: $c_v = \frac{\sigma}{\mu}$. Where μ is the arithmetic mean of any quantifiable attribute (in our case: the stock) and σ is the standard deviation which describes the average deviation of data values from the mean-value:

$$\sigma(x_1, x_2, ..., x_n) = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

[6, p. 81, 7, p. 107]

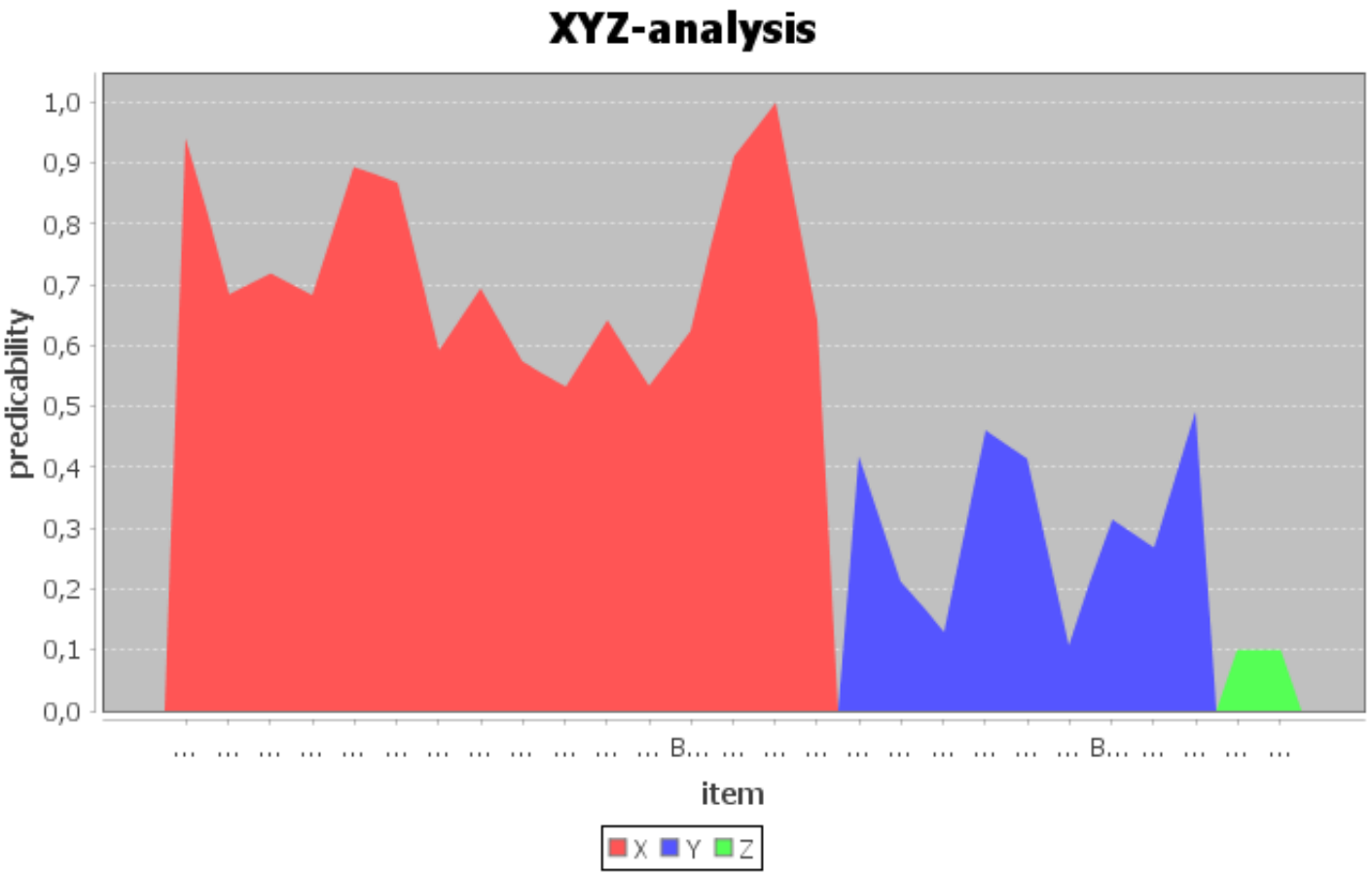

Figure 12: Screenshot from DynaMO, showing an XYZ analysis with sample data. The y-axis shows a predictability relative to to predictability to all other items. In this sample case, the better part of all items are well predictable. This is due to the data being randomly generated, allowing for a generally low deviation. The parameters here were X=0.0, Y=0.2, Z=0.8.

2.4.3 *ABC/XYZ Analysis*

The results of the above views can be combined into the so called *ABC/XYZ analysis*, pairing each category of the ABC analysis, with each category of the XYZ analysis, forming a 3D-table. Consider two items, I_1 and I_2: while both items may fall into category A in the ABC analysis, I_1 could be in category X in the XYZ analysis, while I_2 lies in category Z, making I_2 more interesting for optimisation.

2.4.4 *Reach Analysis*

The *reach analysis* (also known as *inventory coverage*) describes for how long the current stock of an article, a group of articles or the whole inventory will last, considering their average consumption rate. It is therefore calculated as the ratio of the current item stock to the average consumption rate of the item over time.
Generating a reach analysis can be used to determine an ordering date for articles to ensure just-in-time delivery and is therefore tightly coupled with the ABC/XYZ analysis as especially X-items are predestined for this view.
A reach analysis can also visualise a safety stock for the analysed item. Should the minimal actual stock for an item always by far exceed the safety stock, the ordering-frequency or quantity should be adjusted accordingly. In case of DynaMO, the reach analysis is paired with an overview of in- and out-going movements of an article, allowing for a visual analysis of the stock.

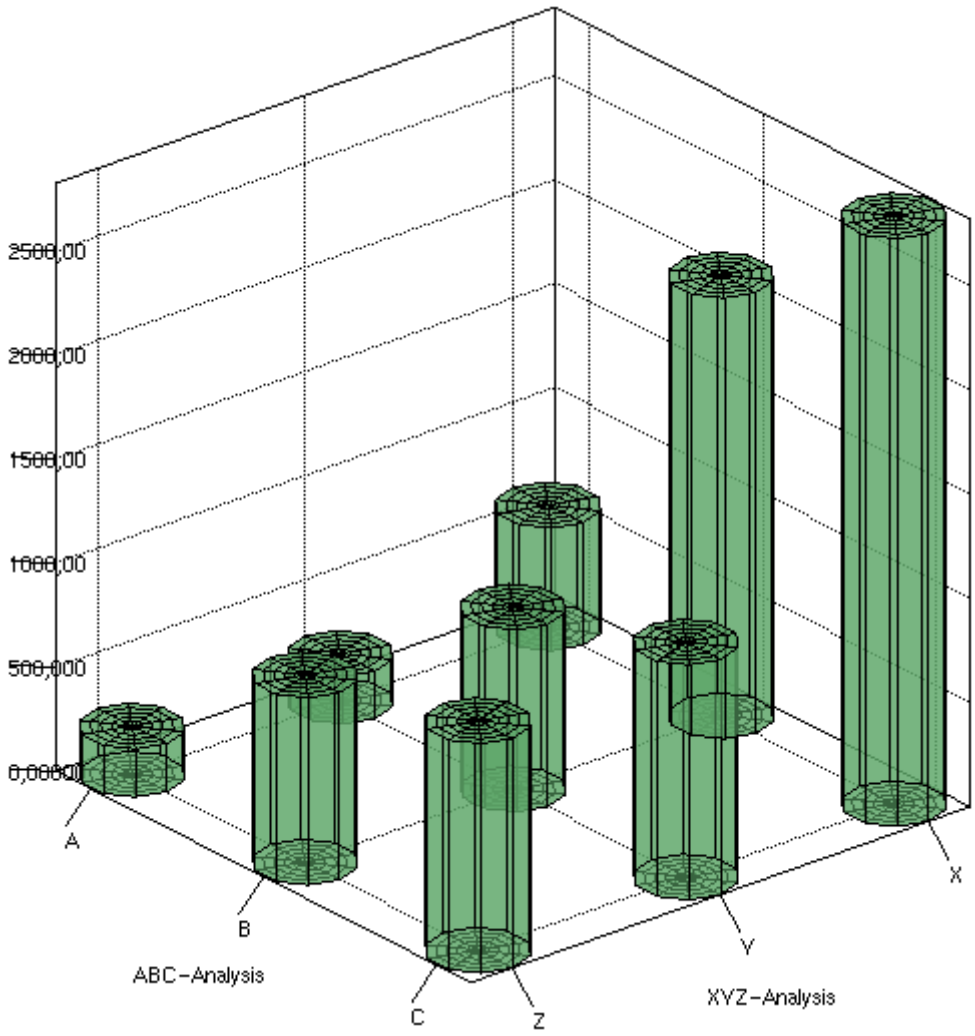

Figure 13: Screenshot from DynaMO, showing an ABC/XYZ analysis of randomly generated data. Items that fall into the Z/A category should be analysed further as those are unpredictable in their usage, but make up a fair amount of the total stock value.

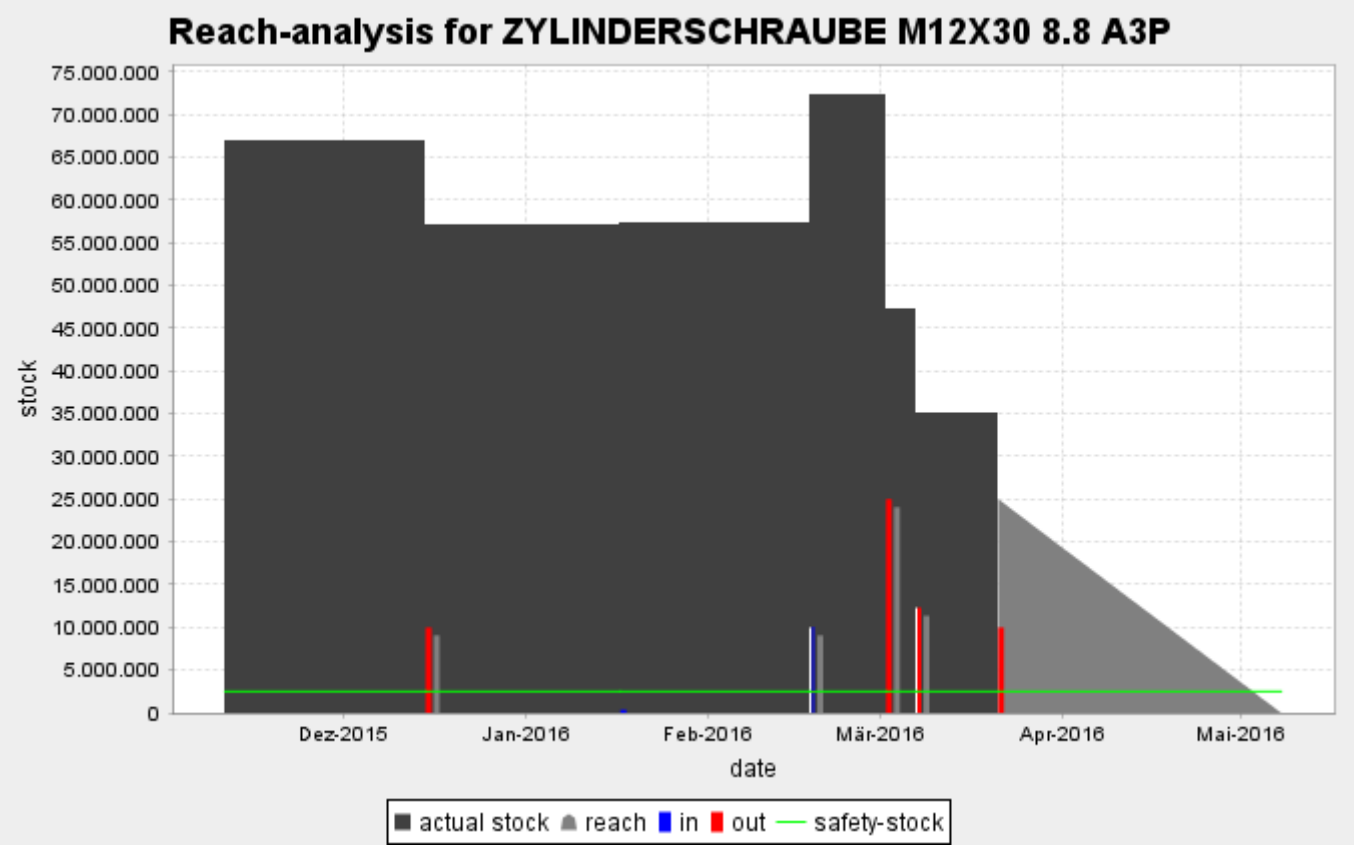

Figure 14: Screenshot from DynaMO, showing a reach analysis over a sample item reconstructed from it's movement history (dark grey). Also displayed is a constant safety stock (green), defined by the user. Obviously leaving some room for improvement as the actual stock never touches the safety stock. The plot also includes a prognosis for how long the current stock will probably last (light grey). The intersection between safety stock and reach therefore marks the latest date, when make-up supplies should arrive.

2.4.5 *Expert Analysis*

The expert analysis has been described in detail in Section 1.2.2.1. DynaMO is able to instantiate pre-compiled Java classes containing the definition of a rule at runtime. Each rule represents one heuristic a human expert would apply. This allows the user to dynamically specify a subset of provided rules or write his own rules and add them to the decision process. When a rule is applied, it can either generate a suggestion by creating a new revision of the considered item, or it can enrich a context for additional information the moderator also keeps track of. This context contains meta-information, such as an ABC analysis for the whole inventory, and is accessible by all rules. It therefore assumes the role of the blackboard from the blackboard architecture, as it serves as a medium to communicate between rules beyond the scope of their current state. This allows us to combine the analysis methods described in the preceding sections to emulate the behaviour of an actual human expert: the context contains intermediate results of the stock as a whole, instead of just judging a single item. If possible rules should also contain a human-readable description of what they suggest and specify why they are suggesting it, making it easier for the user to interpret the output.

As a sample, a small set of rules has been created as depicted in Figure 15. An exemplary implementation of one rule can be seen in Figure 16.

```
RULE_1(x, context):
IF:   ABC/XYZ analysis not present
THEN: add ABC/XYZ analysis to context

RULE_2(x, context):
IF:   reach analysis for x not present
THEN: create reach analysis for x

RULE_3(x, context):
IF:   x has movements (having at least ten
      sets the interest to 1.0)
      and the safety stock is undercut at some point
THEN:  re-evaluate the safety stock

RULE_4(x,context):
IF:   ABC/XYZ analysis is present
      AND x falls into ZA-category
THEN: divest x to satisfy the largest outgoing
      movement x had in the past, or the safety stock

RULE_5(x,context):
IF:   reach analysis for x is present
      and x has suppliers
THEN: find the latest possible order date for
      the cheapest supplier to receive the next delivery
      before the safety stock is undercut
```

Figure 15: A small set of rules used by DynaMO.

```
1  public class SafetyStockRule extends DynamoRule {
2     public float interest(final SAPDataRecord item, final
          Context context) {
3        final int movements = item.getMovements().size();
4        float interest = 0;
5        if (movements > 0) {
6           interest = 0.5f;
7        }
8        if (movements > 10) {
9           interest = 1.0f;
10       }
11       return interest;
12    }
13
14
15    public SAPDataRecord apply(final SAPDataRecord item,
          final Context context) {
16       double overcut = 0;
17       double undercut = 0;
18       final double safety = item.getTrunk().getSafetyStock
             ();
19
20       double amount = item.getStock().getAmount();
21       for (final SAPMovement m : item.getMovements()) {
22          switch (m.getDirection()) {
23          case IN:
24             amount -= m.getAmount();
25             break;
26          case OUT:
27             amount += m.getAmount();
28             break;
29          }
30          if (amount < safety) {
31             undercut = Math.max(safety - amount, undercut);
32          } else if (amount > safety) {
33             overcut = Math.max(amount - safety, overcut);
34          }
35       }
36       SAPDataRecord clone = (SAPDataRecord) item.clone();
37
38       if (undercut > 0) {
39          _action = "re-evaluate order date";
40          _reason = "stock fell below the current safety
                stock at least once";
41       } else if (overcut > 0) {
42          _action = "re-evaluate order date";
43          _reason = "stock never touched the current safety
                stock";
44       }
45       return clone;
46    }
47 }
```

Figure 16: RULE_3 from Figure 15, implemented in Java. The methods `interest` and `apply` reflect the condition (IF) and the action (THEN) of the rule respectively.

Figure 17: Screenshot from DynaMO, showing an expert analysis.

3

PROSPECTS

3.1 FUTURE WORK

While FARAD offers all intended functionality, some areas that could be improved have been identified over the course of the development that would go beyond the scope of this thesis:

IMPROVING THE EXPERT SYSTEM Figure 15 shows a small set of rules that is implemented in the context of DynaMO. Since we expect data to provide at least very basic information as described in Section 2.2.1, it may be possible to determine an (at least small) set of rules which are generally applicable and can therefore be part of the framework itself.
The ES also relies on a user to select the best path through the decision tree from all generated paths from leaf- to root-level (which dictates what actions will be taken to improve the stock). By employing a machine learning mechanism, the system could cut off paths prematurely or even reuse the selected path from one run to improve the following runs.

SANITISATION Another important topic is data sanitisation and cross-checking. Many ERPs provide redundant information concerning the managed resources, such as a total stock and separate stock for each item, that could be used to verify the correctness of the total stock. Sanitisation also includes more basic validation, concerning the data-type and -formatting of ERP information to detect poorly maintained data and possibly fix it on the fly, or at least exclude it from the running analysis.

MORE ANALYSIS METHODS Alongside with the additional rules, additional analyses that can be used on general data, could be implemented, allowing for more out-of-the-box functionality when implementing a tool based on FARAD. This includes meta analyses, preceding the actual analysis to detect anomalies. For example: some companies practise so called "booking rituals", where articles are purchased in bundles. The XYZ analysis benefits greatly from detecting those rituals, effectively eliminating false positives.

EXPERT INPUT While FARAD offers a pretty straightforward way to write new rules as illustrated in Figure 16, doing so still requires the author to have at least a basic understanding of the Java programming language. We would therefore like to get rid of the Java-specific syntax and enable the user to write rules in a language that comes more natural to laymen. As the framework already permits including externally compiled Java-classes as rules at runtime, a *domain-specific language* (DSL) would come in handy to allow users to specify new rules in a simplified language, which is then translated to includable

```
FooRule OPERATES ON BarData

CONDITION:
    ...
ACTION:
    ...
EXPLANATION: "..."  BECAUSE "..."
```

Figure 18: Possible structure of one rule in a DSL for the framework, loosely following the syntax Pascal languages to be closer to natural language. Each rule would have to specify four sections:

1. the type of data the rule operates on
2. the condition (interest-method)
3. the action that is being taken (apply-method)
4. a human-readable explanation.

The code parts could either be implemented in a full-fledged language, contain embedded Java code or both.

source code. Ideally, rules would syntactically resemble the schematic in Figure 3, while still retaining the flexibility of Java. An exemplary structure is proposed in Figure 18, containing all parts of a rule that are relevant for the framework.

3.2 CONCLUSION

The presented framework provides the user with basic analysis- and visualisation-methods, that are easily extendible and adjustable for arbitrary ERPs. The underlying ES is flexible enough to cater to any source systems and offers a general mechanism to evaluate input. All components of the framework are interchangeable which allows developers to use any technology that is suitable to satisfy the requirements of the specific ERP they are evaluating (database systems, different file formats, other evaluation methods than an ES, etc.).
Seeing that, FARAD offers a suitable approach to analyse ERPs on the fly, while still leaving room for improvement beyond the scope of this thesis.

BIBLIOGRAPHY

[1] Armin B. Cremers. *Expertensysteme für die Planung der Produktion*. Köln: Verl. TÜV Rheinland, 1991, 170 S. ISBN: 3-88585-974-2. URL: http://digitool.hbz-nrw.de:1801/webclient/DeliveryManager?pid=1103317&custom_att_2=simple_viewer.

[2] Michael Karst. "Methodische Entwicklung von Expertensystemen". Hochschulschrift. Wiesbaden, 1992, XX, 271 S. ISBN: 3-8244-0106-1. URL: http://swbplus.bsz-bw.de/bsz029199719cov.htm.

[3] John Durkin. "1 - History and applications". In: *Expert Systems*. Ed. by Cornelius T. Leondes. Burlington: Academic Press, 2002, pp. 1 –22. ISBN: 978-0-12-443880-4. DOI: http://dx.doi.org/10.1016/B978-012443880-4/50045-4. URL: http://www.sciencedirect.com/science/article/pii/B9780124438804500454.

[4] John Durkin. "2 - Tools and applications". In: *Expert Systems*. Ed. by Cornelius T. Leondes. Burlington: Academic Press, 2002, pp. 23 –52. ISBN: 978-0-12-443880-4. DOI: http://dx.doi.org/10.1016/B978-012443880-4/50046-6. URL: http://www.sciencedirect.com/science/article/pii/B9780124438804500466.

[5] Arnab Chatterjee. *Econophysics of Wealth Distributions: Econophys-Kolkata I*. Ed. by Sudhakar Yarlagadda and Bikas K. Chakrabarti. Milano: Springer Berlin Heidelberg, 2005, Online–Ressource (X, 248 p, online resource). ISBN: 978-884-7003-89-7. URL: http://nbn-resolving.de/urn/resolver.pl?urn=10.1007/88-470-0389-X.

[6] Wolfgang Kohn. *Statistik: Datenanalyse und Wahrscheinlichkeitsrechnung*. Berlin; Heidelberg [u.a.]: Springer Berlin Heidelberg, 2005, XVI, 622 S. ISBN: 3-540-21677-4. URL: http://swbplus.bsz-bw.de/bsz113245211inh.htm.

[7] Dawn Griffiths. *Statistik von Kopf bis Fuß*. Beijing; Köln [u.a.]: O'Reilly, 2009, XXXV, 678 S. ISBN: 978-3-89721-891-8. URL: http://swbplus.bsz-bw.de/bsz305157876cov.htm.

[8] J. Hilmar Vollmuth. *Controllinginstrumente*. HaufeLexware GmbH & Co. KG, Munzinger Straße 9, 79111 Freiburg: Haufe, 2011. ISBN: 9783648014219. URL: https://www.wiso-net.de/document/HAUF__9783648014219127.

[9] Andreas Huber and Klaus Laverentz. *Logistik*. München: Franz Vahlen, 2012, Online–Ressource (XIV, 247 S.) ISBN: 978-3-8006-4183-3. URL: http://dx.doi.org/10.15358/9783800641833.

[10] Christian Schawel. *Top 100 Management Tools: Das wichtigste Buch eines Managers Von ABC-Analyse bis Zielvereinbarung*. Ed. by Fabian Billing. 5., überarb. Aufl. 2014. Wiesbaden: Gabler Verlag, 2014, Online–Ressource (XIV, 321 S. 170 Abb, online resource). ISBN: 978-383-49469-1-1. URL: http://nbn-resolving.de/urn/resolver.pl?urn=10.1007/978-3-8349-4691-1.

[11] *Genetic Query Optimisation (GEQO) in PostgreSQL*. http://www.postgresql.org/docs/9.2/static/geqo-pg-intro.html. [Online, accessed 2016-02-23].