Abstract—Menu construction is an important task for institutions that need to plan menus within certain constraints. There is also a personal need for professional menu construction by clients or patients who should eat according to a planned diet. For menu construction and dietary analysis, there are several approaches (e.g., linear programming, genetic algorithms, rule-based expert systems, etc.) and commercial IT systems. In this paper, we propose a case-based approach for diet recommendation. Based on this approach, we are going to construct an expert system which is intended to be employed in a health profile management system. Our approach is based on ripple down rules (RDR), however, a special representation is also needed for patient attributes and rule actions.

I. INTRODUCTION

Menu construction is an important task for schools, hospitals, nursing homes, etc., since such institutions need to plan menus within certain constraints such as available material, equipment, and financial means. There is also a personal need for professional menu construction by clients or patients who should eat according to a planned diet, due to certain reasons such as medical conditions or desire to become or stay healthy.

Numerous IT systems and approaches are available today for dietetic applications, and, primarily, for menu construction and dietary analysis. There are several approaches based on linear programming [1]–[3], genetic algorithms [4], [5], rule-based or case-based expert systems [6], [7]. A lot of commercial menu planning systems exist, and are available [8]–[12].

In this paper, we propose a case-based approach for diet recommendation. Based on this approach, we are going to construct an expert system which is intended to be employed in a health profile management system, funded by the project eFilter¹. The goal of the project is to develop an IT system which filters the collection of available food products, considering the data stored in the health profiles of clients or patients. Health profile includes information about food intolerance, allergy, and diet. An important module of this system is going to implement menu planning. In Section II, we suggest an approach for case-based knowledge acquisition, namely the ripple down rules (RDR). In Section III, we propose a representation of rules for the RDR approach within our expert system, i.e., we discuss patient attributes and rule actions. In Section IV, we focus on implementation.

II. KNOWLEDGE ACQUISITION

Knowledge acquisition seems to be the bottleneck in expert system development. Two experts have to be involved, a domain expert and a knowledge engineer (both highly paid specialists). They need to work together for extended periods to develop the knowledge base.

By applying case-based reasoning systems, we get an approach which does not require a knowledge engineer during the knowledge acquisition process.

A. Case-based Reasoning

The basic idea of case-based reasoning is to solve new problems by remembering solutions to problems which are similar to the current problem. Of course, it is domain-specific which problems are considered similar to each other. To be more precise, the domain expert is the one who can make such a decision. As it will be mentioned in Section II-B, ripple down rules provide a very natural interface for the domain expert to specify the appropriate similarity relation, case by case.

A major practical advantage of case-based reasoning is the fact that domain experts cannot usually specify explicitly why a case is similar to another case, but they rather refer to cases they have encountered during their careers. That is, by using case-based reasoning, domain experts are not forced to provide abstract general rules of what they do.

The major problems of case-based reasoning systems are

1) how to retrieve relevant cases, and
2) how to adapt them to fit a new problem.

B. Ripple Down Rules

By using ripple down rules (RDR) [13], the object space (i.e., the sum of all the object classes) is to be incrementally subdivided into smaller and smaller partitions. The goal is to classify all objects from the same class to the same partition. The process of the above-mentioned subdivision is very simple. Assume an object \( x \) which has been classified incorrectly, into the partition \( p \). In such a case, the partition \( p \) needs to be subdivided into two partitions \( p_1 \) and \( p_2 \), such that the partition to which \( x \) belongs classifies \( x \) correctly. The domain expert himself/herself can provide such a subdivision competently.

¹Participants: WIT-SYS Consulting Co. Ltd. (Budapest, Hungary), Eszterházy Károly College (Eger, Hungary), University of Miskolc (Miskolc, Hungary)
with minimal effort. In RDR, the domain expert is merely required to formulate an explanation or criteria why \( x \) differs from the object \( x_p \), which has previously led to the creation of the partition \( p \).

For a domain expert, it is usually quite easy to provide such an explanation, since it is not much different to explaining their decision to colleagues. The basic idea of RDR is to allow the domain expert to directly construct the knowledge base and to incrementally add rules to it.

Using these explanations, the system builds an RDR tree. Such a tree can be seen in Figure 1. In each node of the tree a logical condition and a conclusion (i.e., a partition) is stored. Each node has at most two children. To one of the children an “except”/“true” link is drawn, and an “else”/“false” link to the other child node.

The classification algorithm checks the condition of subsequent nodes. Let \( C \) denote the condition and \( P \) the partition of the current node. If \( C \) is satisfied then the algorithm classifies the input into \( P \). However, this is not necessarily the final verdict, since the node may have an outgoing “except”/“true” link. It has such a link then the algorithm walks along this link, and the classification process continues; otherwise, the classification ends (i.e., the last verdict gets accepted).

Something different happens when \( C \) is not satisfied; the algorithm checks if the node has an outgoing “else”/“false” link. If it does then the algorithm walks along this link, and the classification process continues; otherwise, the classification ends.

For instance, using the RDR tree and the input case that can be seen in Figure 1, the input is first classified into the partition \( P \), by Rule 1. However, this is not the final verdict since Rule 1 has an outgoing “true” link. Since the condition of Rule 2 is also satisfied, the input is temporarily classified into the partition \( Q \). Since the condition of Rule 3 is not satisfied, the algorithm follows the “false” link. The condition of Rule 5 is not satisfied, hence, the classification process ends, and the system adds a new node (i.e., rule) into the RDR tree. The condition of this node is based on the justification that the domain expert has provided.

### III. KNOWLEDGE REPRESENTATION

As it has been proposed, we represent the dietetic knowledge as an RDR tree. However, we should decide how to deal with the following questions.

- What attributes to use for the cases?
- What kind of conditions to use in the nodes?
- What kind of partitions to use?

#### A. Attributes

Attributes can be numerical, string, or boolean values. For example, \( \text{weight} \) is a numerical attribute. The \( \text{lifestyle} \) of a patient could be a string, with such values as “sedentary” or “critically ill”. A boolean attribute \( \text{hasDiabetes} \) could designate whether the patient suffers from diabetes.

In the health profile management system by eFilter, the following attributes describe a patient:

- \( \text{height} \) – integer
- \( \text{weight} \) – integer
- \( \text{bmi} \) – body mass index (BMI), automatically calculated from \( \text{height} \) and \( \text{weight} \)
- \( \text{age} \) – floating-point number
- \( \text{sex} \) – string or enumerated (“male”, “female”)
- \( \text{lifestyle} \) – string or enumerated (“sedentary”, “sporty”, “critically ill”, “handicapped”)
- \( \text{hasDiabetes} \) – This is not an attribute, but rather a function, which takes the name of a disease as a parameter, and returns a boolean value.

#### B. Conditions

Rules’ conditions are to be formulated on the above-mentioned attributes. For instance, one can restrict that \( \text{weight} \) must be not less than 80 kilograms (\( \text{weight} \geq 80 \)), \( \text{lifestyle} \) must be “critically ill” (\( \text{lifestyle} = \text{"critically ill"} \)), or it is to refer only those patients who suffer from protein intolerance (\( \text{hasDisease} \)\("\text{protein intolerance}"\)).

Another question is whether all logical connectives should be permitted in such conditions, or only a few of them? For the sake of the simplicity of the user interface (which is going to be used by a dietitian), we recommend only to use conjunction (i.e., logical “and”) for connecting the atomic conditions.

#### C. Partitions

The most delicate question is how to represent partitions? As a matter of fact, what kind of object or collection of objects should be regarded as a partition, in this dietetic problem? We propose an approach that is similar to the one in [14], hence each partition is represented as a set of actions. Three types of actions are distinguished:

- **Add-action**. The action adds a new constraint on a certain component (e.g., carbohydrate, fat, or protein) [15]. Such a constraint must specify the following data:
Let us assume a patient who is an overweighted boy. For him, the system recommends to consume about 56 grams of protein and 200 grams of carbohydrate daily, and not to exceed 20 grams of fat. However, the dietitian, who is using the system, is not satisfied by this result, since he/she knows that the patient actually suffers from diabetes, thus, he should consume less carbohydrate [17]. This is why the dietitian adds a new rule to the system, by specifying a new condition (hasDisease("diabetes type 2")) and a new replace-action ("carbohydrate" "not recommended" 130). The new rule gets added as can be seen in Figure 3.

IV. IMPLEMENTATION

For implementing our approach, we are to employ open source RDR tools. Let us introduce two such engines in the subsequent sections.

A. BIKE

The Ballarat Incremental Knowledge Engine (BIKE) [18], [19] is a knowledge engineering platform which is designed specifically around the ripple down rules (RDR) family of methodologies. The system can also be extended to other approaches to knowledge engineering. The first release of BIKE was developed at the University of Ballarat, Australia, in 2008–2010. Being a free software, BIKE is available to researchers, developers, students, and the general public through the GNU Affero General Public License.

BIKE does not currently provide either a user interface or any visualizing tool, hence, one can rate BIKE as a poor knowledge engineering software, as compared to more complex systems like WEKA, which we are going to introduce in the next section. However, BIKE provides a C++ API for developers to build sophisticated and easily maintainable RDR-based softwares.

B. WEKA

The Waikato Environment for Knowledge Analysis (WEKA) [20], [21] is a collection of machine learning algorithms for data mining tasks. Its purpose is to collect together learning schemes for a comparative study on a collection of datasets. WEKA has modular, extensible architecture, plugin mechanism, and Java API. It provides tools for data preprocessing, classification, and visualization.

The WEKA project has been funded by the New Zealand government from 1993 up until recently. The first public release was made in 1996, and included 8 learning algorithms and data preprocessing tools written in C. Later, the whole system got rewritten entirely in Java. WEKA is a free software issued under the GNU General Public License.

As already mentioned, WEKA has modular architecture. Its core directly supports multi-instance learning problems, and allows individual algorithms and filters to declare own data characteristics. A lot of learning algorithms have been added to WEKA, e.g.,

- naive Bayes classifiers,
- decision trees,
- neural networks,
- support vector machines (SVN),
- ripple down rules (RDR),
- etc.

The WEKA implementation of a ripple down rule learner is called Ridor. It generates a default rule (i.e., node) at first, and then the exceptions (i.e., "except"/"true" links) with the least error rate. Let us introduce how to use the Ridor classifier by applying a built-in dataset which contains patient data for diagnosis of diabetes. 8 floating-point attributes are stored for each person:

- preg – number of times pregnant
- plas – plasma glucose concentration a 2 hours in an oral glucose tolerance test
- pres – diastolic blood pressure (mmHg)
- skin – triceps skin fold thickness (mm)
Fig. 2. Dietetic knowledge as an RDR tree.

Fig. 3. Dietetic knowledge as an RDR tree – New rule added.

- insu – 2-hour serum insulin (U/ml)
- mass – body mass index (BMI)
- pedi – diabetes pedigree function
- age – age (years)

Furthermore, there is a 9th attribute called class which shows whether the certain patient is diagnosed positive or negative for diabetes. Hence, class is a boolean attribute. After opening the dataset, it may be preprocessed, as can be seen in Figure 4.

WEKA provides a lot of tools for visualizing, just like the one that can be seen in Figure 5, which shows the correspondence between the certain attributes and the diagnosis result.

After preprocessing, we are to begin the classification process. This time, we select the Ridor classifier, as can be seen in Figure 6.

The result of such a classification can be seen in Figure 7. Note that the resulting RDR tree consists of 4 rules (including the default rule).

Fig. 4. WEKA – Preprocessing a dataset.

V. CONCLUSION

Several problems have to be solved during the designing, specifying, and implementing of an IT system for menu construction and dietary analysis. We have focused on diet
recommendation, and proposed a case-based approach. Using ripple down rules (RDR) and suitable rule representation, our approach is going to be applicable in the health profile management system funded by and developed within the scope of the project eFilter.

REFERENCES


