Representation of Medical Guidelines Using a Classification-Based System

C. Heinlein[1] *K. Kuhn*[2] *P. Dadam*[1]
[1]Dept. Databases and Information Systems
[2]Dept. Internal Medicine
University of Ulm, D–89069 Ulm, Germany
{heinlein,dadam}@informatik.uni-ulm.de
kuhn_k@dulruu51.bitnet

Abstract

Medical guidelines play an increasing role in selecting diagnostic and therapeutic steps under the aspects of effectiveness, invasiveness, and costs. To work directly on patient data already available in electronic form, they should be integrated into a medical information system. In order to develop a "medical guideline module" (MGM) managing and applying guidelines to patients, a "knowledge level" representation of guidelines is necessary which reflects the structure of medical knowledge and matches medical processes. Furthermore, a direct transformation to the "symbol level" is needed. We use a nested, frame-like structure on the knowledge level and show that a classification-based knowledge representation system (CBKRS) is principally well suited for the symbol level. To facilitate the usage and to be independent of a particular CBKRS, we introduce an intermediate level called "intelligent object system" (IOS). It is developed by augmenting a simple data model for describing complex objects with *prototypes* and *implications* as a means to classify objects and to draw inferences based on this classification. Finally, the transformation of guidelines to prototypes and implications is described.

1 Introduction

Medicine of today is confronted with exploding costs, exponentially growing knowledge, high specialization, and high informational needs. Increasingly, medical guidelines are used to select adequate diagnostic and therapeutic steps under the aspects of effectiveness, invasiveness, and costs. These guidelines are intended to bring recent scientific results from the literature and from expert physicians into medical practice. For given medical problems of a patient, they suggest necessary ("indicated") steps, warn about side-effects or contra-indications, and remind of preparatory measures. To work directly on patient data already available in electronic form, they should be integrated into a medical information system. The consultation style should be variable from active guidance for unexperienced users to merely issueing certain warnings for experienced physicians.

In: *Proc. Third International Conference on Information and Knowledge Management (CIKM)*, Gaithersburg MD, N. R. Adam, B. K. Bhargava, Y. Yesha (eds.), ACM Press, New York, 1994, pp. 415–422.

In this paper we describe the design and prototypical implementation of a "medical guideline module" (MGM) for managing and applying guidelines to patients, which is part of a medical information system. Major challenges in developing this MGM have been:

- finding an abstract ("knowledge level") representation of medical guidelines reflecting the typical structure of medical information and knowledge; it should be reusable, extensible, and naturally understood by expert physicians;
- identifying essential (abstract) inference mechanisms matching medical processes;
- establishing a direct transformation to a concrete ("symbol level") representation.

Section 2 describes the structure of medical guidelines and an abstract representation by means of nested frames, as well as the basic inferences necessary to apply guidelines to patients. Section 3 identifies these inferences as classification problems naturally being solved by a classification-based knowledge representation system (CBKRS). It also identifies the essential CBKRS features needed by the MGM and defines a corresponding functional interface called "intelligent object system" (IOS) which abstracts from a particular CBKRS and provides a small and uniform interface.

Section 4 describes the transformation of abstract, frame-structured guidelines to concrete "symbols" of the IOS and demonstrates their application to actual patients. In section 5 we will discuss related and future work.

2 Abstract Representation and Inferences

Medical Guidelines

The complexity and general structure of the medical guidelines we consider can be illustrated with the following example. A specific cause for the common disease arterial hypertension (elevated arterial blood pressure) can be identified in a small minority of patients only, but a patient may be cured if this cause is detected. To rule out an endocrine disease as one of these causes, a series of diagnostic steps should be performed. Typically, results determine subsequent steps, and it would be uneconomic or too invasive to perform all tests at a time. So, basically, only potassium and creatinine will be determined from a blood sample, and an ultrasonography examination of the kidney region will be performed. If e. g. hypokalemia (decreased potassium in the patient's blood) is

found, the suspicion of primary aldosteronism (which can be caused by a tumor of the suprarenal gland) will be risen, and the aldosterol and renin serum/plasma levels should be determined. Next steps—again depending on the outcomes—may follow and finally lead to a surgical intervention.

This example is translated into the frame-like structure shown in figure 1 in a straight-forward and natural way. The essential parts of a frame are conjunctive *conditions* (disjunction and negation will be discussed in section 5), and *steps* indicated when the conditions are satisfied. Additional slots, such as a short explanation of the steps or references to literature explaining in more detail the indication, might be included. Frames can be nested in order to represent a context binding implicit in many guidelines. For example, from the condition "potassium < 3.5" the step "test of aldosterol and renin" is derived only in the context of the problem "arterial hypertension." The introduction of subframes inheriting the conditions of their superframes is a natural way to represent this context binding.

Formally, conditions and steps consist of a keyword (e. g. *problem*, *result* or *exam*) followed by a one or more terms taken from a controlled vocabulary. The keyword determines the interpretation of the following terms, e. g. "*problem* x" means "patient has problem x," while "*exam* x y . . . " means "examination x with parameters/questions y . . . is indicated."

The construction of a controlled medical vocabulary is discussed extensively in the literature [1, 2, 3], and several important requirements have been identified [4]. The most important property in our context is "multiple classification," allowing a term to specialize one or more other terms. Figure 2 shows an excerpt of an "ad hoc" term hierarchy which has been sufficient for building the MGM prototype. In the long run, however, we intend to replace it by an existing, more sophisticated approach.

Cond.: *problem* art-hyp
Step: *exam* lab-exam potassium creatinine
Step: *exam* ultrasonography . . .
Expl.: basic diagnostic for arterial hypertension

Cond.: *result* lab-exam potassium < 3.5
Step: *exam* lab-exam aldo renin
Expl.: rule out aldosteronism

Cond.: *result* lab-exam aldo increased
Cond.: *result* lab-exam renin decreased
Step: *problem* prim-aldost

⋮

Cond.: *result* lab-exam creatinine > 120
Step: *exam* lab-exam crea-clearance . . .
Expl.: renal hypertension?

⋮

Figure 1: Guideline frame "arterial hypertension" (see figure 2 for an explanation of terms)

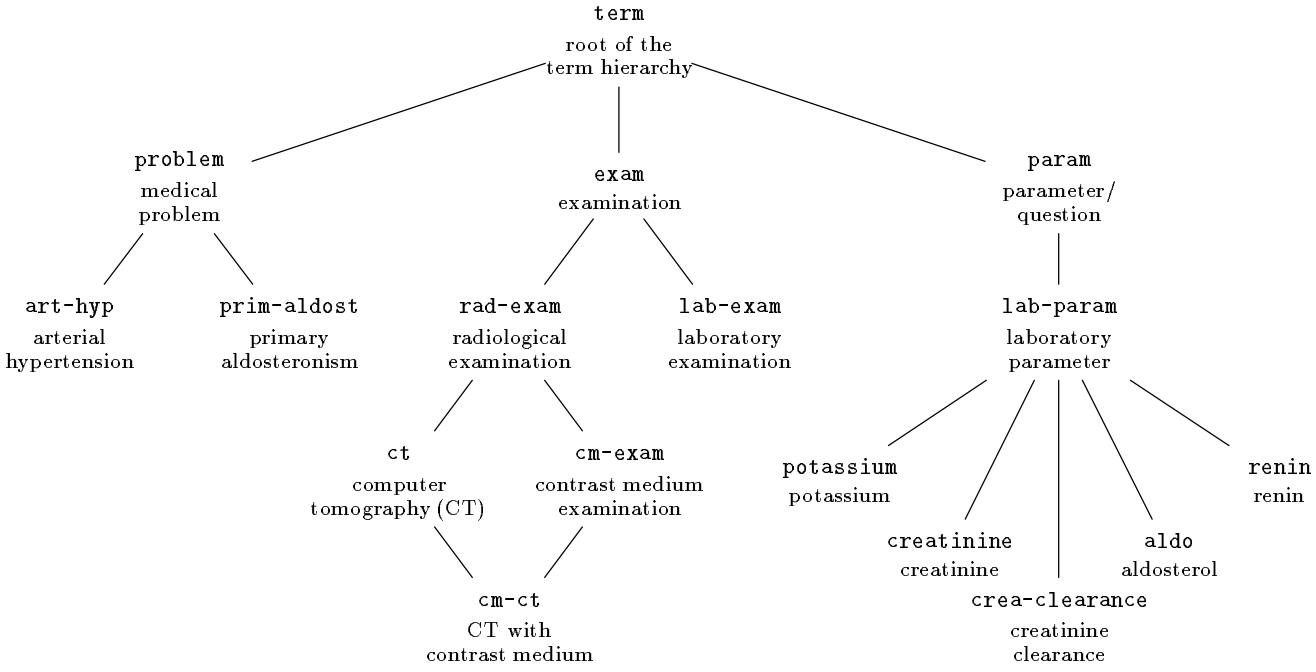

Figure 2: Term hierarchy (links represent specialization)

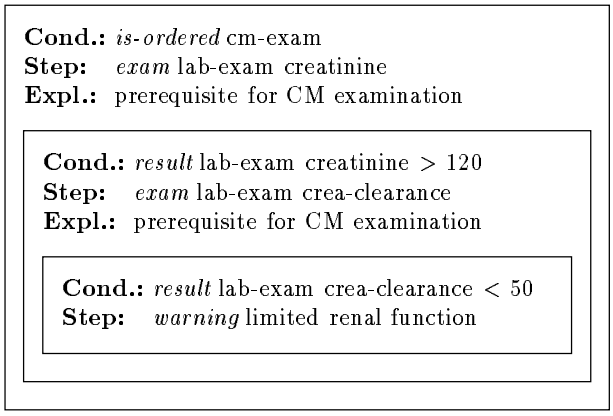

Figure 3: Guideline frame "contrast medium examination"

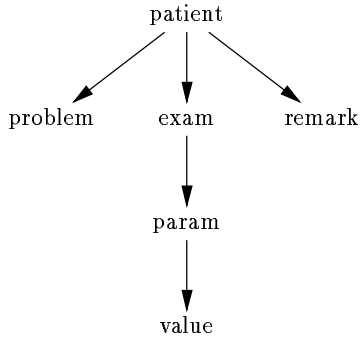

Figure 4: Complex object "patient"

Figure 3 shows an example of a guideline frame making explicit use of this specialization hierarchy. It describes preparations for a radiological examination with application of contrast medium (CM), which may lead to a warning because of limited renal function. As its condition refers to the general term **cm-exam**, it is also satisfied when a more specialized examination such as **cm-ct** has been ordered, and thus need not be reformulated for all these examinations. It might be called a reusable "library" frame.

Patients

Facts about patients might be stored in several different databases of the medical information system. In order to apply guidelines to a patient, his facts must be made available to the MGM in a predefined structure. For that purpose, they are grouped together in a complex object partially shown in figure 4. Arrows denote relationships or *roles* which are viewed as set-valued functions with names derived from their domain and range (e. g. **patient–>exam**). The relationship **patient–>problem** is used to store medical problems, while results of examinations are represented as values (**param–>value**) of parameters/questions (**exam–>param**) of the patient's examinations (**patient–>exam**). Given that representation, conditions of frames can be interpreted more formally than above, e. g. **problem art-hyp** means that the patient object under consideration has a **patient–>problem** role filler of type **art-hyp**, while **result lab-exam potassium < 3.5** means that it has a **patient–>exam** role filler of type **lab-exam** having itself an **exam–>param** role filler of type **potassium** with a **param–>value** role filler that is lower than 3.5.

The same relationships as well as additional ones such as **patient–>remark** will be used by the MGM to store *derived* facts about the patient, as described below.

Inferences

In order to apply guidelines to a given patient, the MGM must determine all frames with conditions satisfied by this patient's facts. The corresponding steps, which may be indicated examinations including parameters/questions, concluded medical problems, or warnings to be issued, must be added to the patient object using the appropriate relationships (e. g. **patient–>remark** for warnings, reminders etc.). Because this process may lead to further satisfied conditions, it must be repeated until a "fixed point" is reached, i. e. no further conditions become satisfied. Every time the patient object is changed, i. e. facts about the patient are added or removed (the latter might happen when a medical problem has been solved), the MGM must recompute (or update accordingly) the derived facts.

A second kind of inferences is needed to answer questions like "Is a particular examination indicated at the moment?" or "Under which circumstances is it indicated?" Because the MGM is intended to operate with incomplete knowledge (guidelines as well as facts about patients may be incomplete), the first question is principally undecidable (the examination might well be indicated without the MGM knowing about it). The second question, however, can be answered at least partially by determining all frames containing this examination as a step, and returning their conditions. If a patient satisfies a significant part of these conditions (defined precisely by the application), a hint might be issued: "If the patient would additionally satisfy condition ..., the examination would be indicated for him."

3 Technical Foundation

Finding all frames with conditions satisfied by a patient's facts can be seen as a classification problem. Each frame defines the class of patients satisfying its conditions, and patients have to be classified according to their facts. Because subframes inherit the conditions of their superframes, the class of patients defined by a subframe is a subclass (i. e. a subset) of the classes defined by its superframes. Therefore, each complex frame (i. e. top-level frame including its direct and indirect subframes) induces a corresponding class hierarchy.

Classification-based knowledge representation systems (CBKRSs) provide a language consisting of *concept forming operators* as a means to describe such class hierarchies, and a specialized reasoner, the *classifier*, which automatically identifies all classes an object belongs to [5]. Typical concept forming operators are *conjunction*, stating that an object belongs to class C iff it belongs to classes $C_1, \ldots, C_n$, and several kinds of *number* and *value restrictions*, stating that an object belongs to class C, iff all or at least/at most k fillers of one of its roles belong to a class C'. Furthermore, these systems usually allow some kind of implications or forward-chaining rules to be associated with a class, stating that an object of class C also belongs to class D. If the definition of D prescribes concrete role fillers for its instances (which

might be considered a special kind of value restriction), these rules can be used to automatically add additional role fillers to objects matching class C. These derived role fillers are truth-maintained, i. e. they are retracted again, if the object no longer satisfies the conditions of class C. This combination of classification, forward-chaining rules, and truth maintenance is exactly what is needed in order to provide the first kind of inferences described above. (The second kind will come as a by-product at the end of section 4.)

A typical CBKRS provides a large set of functions, operators, etc. [6, 7], many of which are rarely needed by an application. Sometimes there exist several different functions for similar purposes, leading to a large and unwieldy interface. Furthermore, different CBKRSs provide quite different interfaces for the same or similar features, causing serious portability problems [8]. Therefore it is reasonable to introduce an intermediate layer which abstracts from a particular CBKRS, concentrates on the essential features needed, and provides a small and uniform interface for them. In the remainder of this section, an intermediate layer of this kind will be developed step by step. After defining a data model for describing complex objects, *prototypical individuals* and *implications* will be introduced as a means to classify objects and to draw inferences based on this classification. Afterwards we will briefly sketch the implementation of this layer called "intelligent object system" (IOS), and describe in more detail the CBKRS features it requires. In section 4, the IOS will be used as a platform to implement the MGM.

Terms, Types, and Relationships

A controlled vocabulary constitutes the basis of every knowledge-based system. In order to build term hierarchies as the one shown in figure 2, we postulate a function **term** taking as arguments a term to be defined and optionally a set of superterms it specializes, for example:[1]

```
(term exam)
(term rad-exam  exam)
(term ct  rad-exam)
(term cm-exam  rad-exam)
(term cm-ct  ct cm-exam)
```

On the one hand, a term is a word, i. e. an element of a vocabulary, while on the other hand it is a *type* representing a class of real-world objects. In order to build complex objects like the one shown in figure 4, the function **relation** is introduced, taking as arguments domain and range of a relationship. For example:

```
(relation patient problem)
(relation patient exam)
(relation exam param)
(relation param value)
(relation patient remark)
```

Individuals

For each type, there exists a predefined individual with the same name as the type. Furthermore, the name of a type can be used as a function ("type function") to create a new individual of that type. Initial role fillers may be specified as a sequence of relationship–individual pairs, e. g.:

```
(patient patient->problem art-hyp)
```

[1] Since almost every CBKRS is based on Lisp, it was a natural decision to implement the IOS in Lisp, too.

This creates a new **patient** individual with the **patient–> problem** relationship filled by the predefined individual **art-hyp**. Because initial role fillers can be created in the same way, a complex individual can be created with a single expression resembling the conceptual graph notation of [9]:

```
(patient
  patient->problem art-hyp
  patient->exam
    (lab-exam exam->param
      (potassium param->value 3.2)
    )
)
```

This creates a patient individual with the problem arterial hypertension and a laboratory finding of potassium with value 3.2.

The function **current**, taking as argument a type, returns the "current" instance of that type, i. e. the last individual created of that type. The name of a relationship can be used as a function ("relationship function") to establish that relationship between two individuals. This allows the same individual to be built incrementally:

```
(patient)
(patient->problem (current patient) art-hyp)
(patient->exam (current patient) (lab-exam))
(exam->param (current lab-exam) (potassium))
(param->value (current potassium) 3.2)
```

A relationship function can also be used to retract role fillers and to query the current set of role fillers for a given individual as in the following example, where all medical problems of the current patient are returned:

```
(patient->problem (current patient))
```

Prototypes

On the one hand, the individual created above might describe a single patient of the real world having the problem arterial hypertension and a potassium value of 3.2. On the other hand it may be viewed as a description of the class of *all* patients having this problem and this potassium value. If it is used in the latter way, it is called a *prototypical individual* or a *prototype*, and every element of the class being described by it is said to *match* that prototype. In order to allow for more general class descriptions, prototypes may have symbolic role fillers like **(< 3.5)** or **(> 120)** to describe *ranges* of values instead of concrete numerical values.

Formally, matching a prototype is defined as follows:

- A type is called a *subtype* of another type, if both types are equal, or if the former is defined as a specialization of the latter.
- A numerical value is said to match a range, if it is an element of that range.
- An individual is said to match a prototype, if the individual's type is a subtype of the prototype's type, and for each role filler of the prototype, there exists a *matching* role filler of the individual.

The last part of the definition has to be recursively applied, if role fillers are themselves individuals or prototypes. Roughly speaking, these definitions say that a prototype describes the class of all individuals being at least as specialized as the prototype itself.

Because prototypes describe classes of individuals, they can be used as query expressions. The function **all** returns the set of all individuals matching a given prototype, e. g.

```
(all (patient patient->problem art-hyp))
```

This returns all patients with the problem arterial hypertension.

The function **copy** creates an exact copy of a prototype (or an arbitrary individual). This is useful for creating "subprototypes," because after adding additional role fillers to the copy, it can be used as a prototype describing a subset of the class described by the original prototype.

Implications

In order to build an *intelligent* object system, we introduce implications of the form: "As long as an individual matches prototype A, it is guaranteed to also match prototype B." In order to satisfy this "guarantee," an individual matching prototype A must automatically receive role fillers matching the role fillers of B in addition to its original role fillers. (This is most easily accomplished by receiving exactly the role fillers of B.) As the term "as long as" indicates, these additional role fillers must be retracted again, if the individual does no longer match prototype A.

The function **implication** taking as arguments two prototypes, is used to define such an implication, e. g.

```
(implication
  (patient patient->problem art-hyp)
  (patient
    patient->exam
      (lab-exam
        exam->param potassium
        exam->param creatinine
      )
  )
)
```

This guarantees that every patient having problem **art-hyp** automatically receives a **patient–>exam** role filler of type **lab-exam** having **exam–>param** role fillers with types **potassium** and **creatinine**.

Implementation of the IOS

Terms, relationships, and individuals (including prototypes) are naturally mapped to *primitive concepts, relations* and *instances* of a CBKRS, respectively. In order to build term hierarchies, *conjunction* of primitive concepts is needed.

The first argument to **implication** is converted to a *defined concept* describing exactly the set of individuals matching this prototype. The concept forming operators needed here, are *conjunction* and *qualifying existential restriction* (stating that an individual must have at least one role filler of type C' in order to be a member of class C) as well as a means to describe numerical ranges. In the syntax of Loom [6], which has been used for the prototypical implementation, the patient individual shown above would be converted to:

```
(defconcept ... :is
  (:and patient
    (:some patient->problem art-hyp)
    (:some patient->exam
       (:and lab-exam
         (:some exam->param
            (:and potassium
              (= param->value 3.2)))))))
```

The second argument to **implication** is converted to a concept definition prescribing concrete role fillers for its instances, for example:

```
(defconcept ... :is
  (:and patient
    (:filled-by patient->problem prim-aldost)))
```

The implication itself is mapped to an *implication* or *rule* of the CBKRS with the first concept as antecedent and the second as consequent.

A prototype passed to **all** might be converted to a defined concept in the same way as the first argument to **implication**, and afterwards the CBKRS can be queried for all individuals matching that concept. If the CBKRS allows for more complex queries, however, the prototype can be converted directly to an equivalent query expression, and no "auxiliary" concept definition is needed.

4 The Medical Guideline Module

Transformation of Frames to Prototypes and Implications

A "guideline base" is a set of frame-structured guidelines as described in section 2. In order to make them applicable to actual patients, they are transformed to prototypes and implications of the IOS, i. e. eventually to concepts and rules of the underlying CBKRS. For each frame, two prototypical patients are created—one to represent the conditions, and the other to represent the steps of the frame. Because the "condition prototype" must be matched by all patients satisfying the conditions of the frame, each condition is converted to a "prototypical" role filler. For example, the condition **problem art-hyp** is transformed to a **patient–>problem** role filler of type **art-hyp**, while **result lab-exam potassium < 3.5** produces a **patient–>exam** role filler of type **lab-exam** having itself an **exam–>param** role filler of type **potassium** with a **param–>value** of **(< 3.5)**. In the same way, steps are transformed to role fillers of the "step prototype." This transformation—which depends heavily on the structure of the complex patient object (figure 4) as well as the internal structure of conditions and steps—is done by a separate "parsing function." If new keywords shall be introduced for conditions or steps, this is the only function needed to be modified.

After these transformations, an implication is declared between the two prototypes. According to the definition of an implication in the IOS, this guarantees that each patient matching the condition prototype, i. e. satisfying the conditions of the frame, also matches the step prototype, i. e. has the steps of the frame as derived role fillers.

In order to make a subframe inherit the conditions of its superframes, its condition prototype is created as a copy of its parent's condition prototype and augmented with role fillers corresponding to its own conditions.

Suggesting Indicated Steps

In order to apply a guideline base to a given patient, a patient individual is created and filled with all facts available about the patient in the information system. The implications mentioned above cause all indicated steps to be automatically added to the individual. From the CBKRS's point of view, the classifier determines all concepts matched by the individual and applies the associated rules. Whenever the individual is changed, it is reclassified and its derived role

fillers are updated accordingly. Therefore, an application can obtain the indicated steps at any time by querying the role fillers of the individual, e. g. **patient–>exam** to get indicated examinations, **patient–>remark** to get reminders and warnings, and so on.

In order to demonstrate this behavior in more detail, we will briefly sketch an interactive "test session" with the MGM. After loading the CBKRS, IOS, and MGM code, as well as a sample knowledge base containing the frames described here, into a Lisp system, a new, "empty," patient individual is created which is afterwards accessible as the "current" patient:

```
> (patient)
```

In order to tell the system that this patient suffers from arterial hypertension, a corresponding **patient–>problem** role filler is added:

```
> (patient->problem (current patient) art-hyp)
```

After this "assertion," the system recognizes the patient to satisfy the top-level condition of the frame in figure 1, and therefore adds the corresponding steps as derived role fillers. This may be verified by querying the **patient–>exam** role fillers:

```
> (patient->exam (current patient))
(lab-exam--1 ultrasonography--1)
```

Querying the **exam–>param** role fillers of **lab-exam--1** yields:

```
> (exam->param lab-exam--1)
(potassium--1 creatinine--1)
```

In order to inspect a complex object more conveniently, the function **show** can be used to show a piece of code which would exactly reproduce the given individual:

```
> (show (current patient))
(patient
  patient->problem (art-hyp)
  patient->exam
    (lab-exam
      exam->param (potassium)
      exam->param (creatinine)
    )
  patient->exam
    (ultrasonography ...)
)
```

An application-specific function **suggest** may take the derived **patient–>exam** role fillers (i. e. the indicated examinations), filter out those which have already been carried out, and expand the "cryptic" terms into more readable texts. If we assume that no examinations have been performed yet, it would yield:

```
> (suggest (current patient))
* laboratory examination
  (basic diagnostic for arterial hypertension)
  - potassium
  - creatinine
* ultrasonography
  (basic diagnostic for arterial hypertension)
  - ...
```

After having performed the suggested laboratory tests, their results are entered by augmenting the patient individual with a corresponding **patient–>exam** role filler:

```
> (patient->exam (current patient)
    (lab-exam
      exam->param (potassium param->value 3.2)
      exam->param (creatinine param->value 110)
    )
  )
```

According to the guideline in figure 1, potassium is decreased (< 3.5) while creatinine has a normal value. Therefore, the condition of the first inner frame is satisfied, leading to the suggestion of a laboratory test of aldosterol and renin. Since the above ultrasonography has not been performed yet, it is suggested again:

```
> (suggest (current patient))
* ultrasonography
  (basic diagnostic for arterial hypertension)
  - ...
* laboratory examination
  (rule out aldosteronism)
  - aldosterol
  - renin
```

This "dialog" of entering new results and asking for the next indicated steps may be continued that way.

In order to test the system's behavior in case of retractions, too, one might remove the problem **art-hyp** now. Then, the patient no longer satisfies the top-level condition of figure 1 nor the conditions of any subordinate frames since they inherit this top-level condition. Therefore, all derived role fillers will be retracted, causing the output of **suggest** to become empty. The only remaining role filler of the current patient is the laboratory examination which has been explicitly asserted:

```
> (patient->problem (current patient)
    :retract art-hyp)
> (show (current patient))
(patient
  patient->exam
    (lab-exam
      exam->param (potassium param->value 3.2)
      exam->param (creatinine param->value 110)
    )
)
```

Finding Preconditions of a Step

Because each prototype matches itself, the implication associated with a condition prototype is applied to the prototype itself as well, i. e. each condition prototype receives the role fillers of the corresponding step prototype as derived facts. This can be employed to determine the preconditions of a particular step by looking for all condition prototypes containing that step, because their original role fillers constitute exactly these preconditions.

In principal, **all** can be used to obtain all individuals containing that step, and afterwards relationship functions like **patient–>problem** can be used to obtain their role fillers. Two problems remain, however: **all** is unable to distinguish condition prototypes from other individuals, and the relationship functions cannot distinguish between original and derived role fillers. In order to fix these problems, we augment the IOS with two simple partitioning mechanisms which might be useful for other purposes as well. First, type functions such as **patient** as well as the query function **all** accept an optional argument specifying a *partition* in which individuals are created or searched, respectively. This can

be used to create condition prototypes in a separate partition, thus circumventing the first problem. Second, the relationship functions accept an optional argument specifying which kind of role fillers to query: original, derived, or both.

Since a CBKRS internally distinguishes between original and derived facts anyway, the second extension comes almost for free. If multiple partitions of individuals are not directly supported, an IOS internal relationship representing the partition an individual belongs to, can be used to simulate it.

5 Discussion

We have built a medical guideline module on top of a classification-based knowledge representation system. The combination of classification, forward-chaining rules, and truth maintenance of derived facts provides a powerful platform for building knowledge-based systems rather quickly. Several authors describe the advantages of using a semantically well defined terminology as a "backbone" of an intelligent system, which is usually augmented with production rules as an inferential component [10, 11, 12]. Because we do not use *production* rules which cannot be undone once having "fired," but truth-maintained implications, the situation is even better: rules (i. e. frames in our context) can be formulated in a completely declarative way, without needing to have any strategy for execution or conflict resolution in mind. Indeed, rules are not executed in a procedural way, but instead the set of all facts derivable via rules is maintained by the CBKRS. It remains to be evaluated, though, how these powerful mechanisms affect system performance.

In [13], a semantic data model is described which extends the CBKRS languages FL−, KANDOR, and BACK. It provides a single, homogeneous language for defining objects, queries, and views, instead of different languages for data definition and manipulation. Similarly, the CLASSIC system [7] provides identical language constructs for defining concepts and individuals. We follow this approach in the design of the intelligent object system. The same functions/language constructs needed to create (complex) individuals are used to describe queries, classes, and implications. This principle of "re-using" the same constructs for several purposes, combined with the careful selection of the essential CBKRS features needed, has led to an interface consisting of less than ten simple functions. Furthermore, since individuals can be constructed incrementally, query expressions and class descriptions may also be built step by step, which is sometimes more convenient than generating one complex description as a whole.

The idea of "blending classes and instances" in order to simplify language design and usage is also described for the prototype-based object-oriented programming languages Self [14, 15] and Omega [16].

The definition of matching a prototype, given in section 3, is based only on conjunction and qualifying existential restriction. More general value and number restrictions have not been necessary so far, but they may be added if an application-specific need has been identified. We have excluded negation and disjunction so far, because they make things more difficult and thus are omitted from most available CBKRSs. The authors of [17] tell some simple "tricks of the trade" to achieve a limited form of negation, however, which is sufficient in our context: instead of saying e. g. **(not (potassium < 3.5))** or **(not (ultrasonography normal))** we say **(potassium >= 3.5)** or **(ultrasonography pathological)**, and so on. In order to deal with disjunction, we add the following "trick:" if an individual belongs to class C, if it belongs to class A or B, this can be modeled with the implications $A \Rightarrow C$ and $B \Rightarrow C$.

Because CBKRSs as well as the IOS are application-independent tools providing generic language constructs, it is not surprising that knowledge representation using solely these constructs is more like an "encoding" dealing with "symbols" like concepts, relations, or prototypes, than an abstract description of knowledge concentrating on the essential content. Usually, a few syntactic templates have to be repeated again and again (cf. sample knowledge bases provided with some CBKRSs; one of them is described in [17]). Therefore, we vote for an application-specific "knowledge level" representation which is at the same time "natural" enough to be understood by domain experts and formal enough to be "understood" by a software module translating it to the symbol level. We have chosen nested frames for describing medical guidelines for the following reasons. First, the extensible slot structure of frames represents quite naturally the structure of many "units of knowledge" including rules. In this respect, our approach is similar to the Arden Syntax [18] which has been suggested as a standard representation for medical knowledge. Its "medical logic modules" (MLMs) are essentially frame-structured rules, augmented with slots for maintenance, citations, etc. In order to be easily sharable, MLMs are demanded to be independent of each other, which means that their conditions must be *complete*. While this makes sense for simple, "single-stage" units of knowledge being semantically independent, it leads to unnatural and unwieldy formulations in the area of "multi-stage" guidelines since many conditions have to be repeated in order to achieve the context binding described in section 2. Therefore, we allow frames to be nested, i. e. conditions to be complete only in their well-defined context. This does not sacrifice sharability, though, but changes the granularity of sharable units of knowledge from single rules to complete complex frames.

Due its inherently forward-chaining mode of operation [5], a CBKRS needs all facts about an individual in advance in order to find the most specific concepts it belongs to. The classifier does not "intelligently" ask for additional facts which could "improve" the classification. As long as the MGM is used as an experimental prototype subject to changes and improvements, a complete replication of patient-specific facts in the CBKRS is acceptable. For routine use, however, where the MGM must deal with many patients at the same time, this may lead to memory problems, since a CBKRS usually keeps all knowledge in main memory. In addition, consistency and reliability problems can result. Since not all patient-specific facts available in the information system are relevant for the application of guidelines, a more "intelligent" coupling between the information system and the MGM is desirable.

Active values [10] can be employed to automatically query relevant facts on demand, but since few CBKRSs support them, we intend to investigate an alternative approach which might be called "incremental" classification: Initially, only a few "essential" facts (e. g. medical problems) are used to classify a patient, and the result is exploited to selectively ask the information system for additional facts. In the long run, it might even be worthwile to implement the IOS directly on top of a database system without using a

CBKRS at all, in order to avoid the memory and consistency problems mentioned above. To remain open for this possibility has been another motivation for defining a CBKRS-independent intermediate layer which is as small and simple as possible.

Acknowledgement

We want to thank G. Adler for his support, B. Böhm for providing the example, and B. Nebel, D. Rösner, and T. Beuter for their discussions on an earlier version of this paper.
This work is part of the research project "Open Clincial Database and Information System for the Integration of Autonomous Subsystems" (OKIS), which is supported by the State of Baden-Württemberg, Germany.

References

[1] C. A. Goble, A. J. Glowinski, W. A. Nowlan, A. L. Rector. "A Descriptive Semantic Formalism for Medicine" in *Proc. 9th International Conference on Data Engineering.* IEEE Computer Society Press, April 1993, 624–631.

[2] K. E. Campbell, M. A. Musen. "Representation of Clinical Data Using SNOMED III and Conceptual Graphs" in *Proc. 16th Annual Symposium on Computer Applications in Medical Care.* M. E. Frisse (ed.), McGraw-Hill, November 1992, 354–358.

[3] R. H. Baud, A.-M. Rassinoux, J.-R. Scherrer. "Natural Language Processing and Semantical Representation of Medical Texts" *Methods of Information in Medicine* 31 (2) June 1992, 117–125.

[4] J. J. Cimino, G. Hripcsak, S. B. Johnson, P. D. Clayton. "Designing an Introspective, Multipurpose, Controlled Medical Vocabulary" in *Proc. 13th Annual Symposium on Computer Applications in Medical Care.* L. C. Kingsland (ed.), IEEE Computer Society Press, November 1989, 513–518.

[5] R. MacGregor. "The Evolving Technology of Classification-Based Knowledge Representation Systems" in *Principles of Semantic Networks. Explorations in the Representation of Knowledge.* J. F. Sowa (ed.), Morgan Kaufmann Publishers, 1991, 385–400.

[6] D. Brill. *Loom Reference Manual (Version 2.0).* Information Sciences Institute, University of Southern California, December 1993.

[7] L. A. Resnick, A. Borgida, R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, K. C. Zalondek. *CLASSIC Description and Reference Manual for the COMMON LISP Implementation (Version 2.1).* AT&T Bell Laboratories, Murray Hill, NJ, May 1993.

[8] J. Heinsohn, D. Kudenko, B. Nebel, H.-J. Profitlich. *An Empirical Analysis of Terminological Representation Systems.* RR-92-16, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany, 1992.

[9] J. F. Sowa. *Conceptual Structures. Information Processing in Mind and Machine.* Addison-Wesley, 1984.

[10] R. Fikes, T. Kehler. "The Role of Frame-Based Representation in Reasoning" *Communications of the ACM* 28 (9) September 1985, 904–920.

[11] W. Swartout, R. Neches. "The Shifting Terminological Space: An Impediment to Evolvability" in *Proc. National Conference on Artificial Intelligence.* American Association for Artificial Intelligence, 1986, 936–941.

[12] J. Yen, R. Neches, R. MacGregor. "CLASP: Integrating Term Subsumption Systems and Production Systems" *IEEE Transactions on Knowledge and Data Engineering* 3 (1) March 1991, 25–32.

[13] H. W. Beck, S. K. Gala, S. B. Navathe. "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model" in *Proc. 5th International Conference on Data Engineering.* February 1989, 572–581.

[14] D. Ungar, R. B. Smith. "Self: The Power of Simplicity" in *OOPSLA '87 Conference Proceedings*, N. Meyrowitz (ed.), October 1987, 227–242.

[15] C. Chambers, D. Ungar, E. Lee. "An Efficient Implementation of Self, a Dynamically-Typed Object-Oriented Language Based on Prototypes" in *OOPSLA '89 Conference Proceedings*, N. Meyrowitz (ed.), October 1989, 49–70.

[16] G. Blaschek. "Type-Safe Object-Oriented Programming with Prototypes. The Concepts of Omega" *Structured Programming* 12 (4) 1991, 217–225.

[17] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick. "Living with CLASSIC: When and How to Use a KL-ONE-Like Language" in *Principles of Semantic Networks. Explorations in the Representation of Knowledge.* J. F. Sowa (ed.), Morgan Kaufmann Publishers, 1991, 401–456.

[18] G. Hripcsak, P. D. Clayton, T. A. Pryor, P. Haug, O. B. Wigertz, J. Van der Lei. "The Arden Syntax for Medical Logic Modules" in *Proc. 14th Annual Symposium on Computer Applications in Medical Care.* R. A. Miller (ed.), IEEE Computer Society Press, November 1990, 200–204.