Effective Domain-Dependent Reuse in Medical Knowledge Bases

Michel Dojat[a], François Pachet[b]

[a]*Institut National de la Santé et de la Recherche Médicale, INSERM Unité 296, Créteil, France.*

[b]*Laboratoire Formes et Intelligence Artificielle, LAFORIA-IBP, Université Paris 6, Paris, France.*

Published in Comput Biomed Res 28 (1995) 403-432.

Abstract

Knowledge reuse is now a critical issue for most developers of medical knowledge-based systems. As a rule, reuse is addressed from an ambitious knowledge engineering perspective that focuses on reusable *general purpose* knowledge modules, concepts and methods. However, such a general goal fails to take into account the specific aspects of medical practice. From the point of view of the knowledge engineer, whose goal is to capture the specific features and intricacies of a given domain, this approach addresses the wrong level of generality. In this paper, we adopt a more pragmatic viewpoint, introducing the less ambitious goal of "domain-dependent limited reuse", and suggesting effective means of achieving it in practice. In a knowledge representation framework combining objects and production rules, we propose three mechanisms emerging from the combination of object-oriented programming and rule-based programming. We show how these mechanisms contribute to achieve limited reuse and to introduce useful limited variations in medical expertise.

1. INTRODUCTION

An abundant flow of information is available today to the health-care workers. As Gardner points out (21) , it is not realistic to believe that the human mind can integrate all the pieces of the medical decision puzzle to select the best treatment without assistance. Since Mycin, Artificial Intelligence (AI) techniques have generated considerable interest as a tool for clinical problem solving, and a wealth of computer systems for assisting the clinical staff in critical medical decision-making have been designed (9,34,50).

Research into use of AI in medicine has focused mainly on the construction of Medical Knowledge-Based Systems (MKBSs). Unfortunately, only few MKBSs have been evaluated in clinical environments with the goal of validating them for routine use. Consequently, the impact of AI techniques for improving the quality of patient care is still small (9). This paradox has been widely acknowledged, and several lines

of research have been suggested as capable of filling the gap between the many prototypes developed in research laboratories and the development of systems appropriate for bedside use. In particular, Musen (37) insists on the importance of *sharing* and *reuse* as "necessary conditions for building complex systems that the AI in medicine community ultimately hopes to develop".
In the same vein, Ramoni et al. (45) have argued that the development of real world MKBSs requires a fine epistemological analysis of the medical tasks and reasoning involved. This analysis is useful for two different reasons: 1) to provide a framework for acquiring and modeling knowledge and 2) to facilitate sharing and reuse. These requirements lead to the well-known separation (39) between a conceptual model (or *knowledge level*) and the representation paradigms (or *symbol level*) used to implement it. For these purposes, two complementary but distinct categories of knowledge level descriptions have been identified (37): 1) problem-solving methods, or models of "how a particular task is performed", such as models of diagnosis, therapy planning, and patient monitoring; and 2) ontologies, or structural descriptions of types of domain-knowledge objects and expressions, including taxonomic, causal, and time ontologies.

We are currently working on the design of knowledge-based medical systems intended for use in real-life clinical situations. However, rather than trying to identify and to define *general purpose* knowledge modules, concepts, or methods, we have concentrated on achieving some kind of *limited reuse*, within a given domain. This limited reuse is the focus of this paper.

The development of large medical decision support systems requires formalization of expert knowledge prior to its implementation on a computer. This formalization is useful to compare different types of expertise, to conduct in-depth evaluations of the structure of medical decision making and, when possible, to achieve some degree of standardization. This need for standardization has been emphasized by Morris (36), who advocated *strong* standardization of clinical practices in computerized protocols, contending that this is the only means of scientifically evaluating treatments.

Departing from this approach, we support *loose* standardization. As pointed out by Brender and McNair (5), "gold standards" in medicine may vary from place to place depending on factors related to traditions, socio-economic conditions, and politics, rather than science and ethics. Our experience with the acquisition of medical expertise shows how important it is for the clinician to reason using not only a consensual kernel of concepts but also small variations around this kernel, which reflect the specific characteristics of each patient and clinical team (qualifications, skills, culture, or local habits). We believe that this distinction strongly constrains the structure of the MKBS itself. Therefore, we propose to explicitly take this difference into account, both during the process of knowledge acquisition (at the conceptual level) and during actual construction of the knowledge base (at the symbol level). Rather than formalizing medical knowledge at large for purposes of scientific validation, we propose limited reuse as a means of achieving loose standardization in bounded domains.

The remainder of the paper is structured as follows. In Section 2, we define limited reuse and propose ways to achieve it. In Section 3, we define our epistemological position and distinguish between three knowledge layers, i.e., domain, inference, and control. Section 4 describes the combination of objects and rules used to implement our epistemological model. In Section 5, we illustrate effective domain-dependent reuse with examples of mechanical ventilation management. Section 6 gives more details on NéoGanesh, a working prototype for mechanical ventilation designed using our methodology. Finally, we discuss the importance of limited reuse in medical systems and emphasize the benefits of the association of object-oriented paradigm and production rules to achieve it.

2. Limited Reuse

Our research is being conducted in collaboration with an intensive care unit team at the Henri Mondor Hospital (Créteil, France). The medical domain is patient monitoring and, more specifically, ventilator therapy management. In this context, we have accumulated experience in the gradual acquisition and representation of the team's expertise concerning ventilator therapy management.

2.1. Importance of Limited Reuse

This experience shows the importance of reusing our systems in *particular situations* in which an initial corpus of knowledge undergoes several *small variations.* These variations occur in three main situations, which are outlined below.

1) <u>Small variations of expertise according to its *context* of application</u>

The concept of context is vital in medical knowledge and appears at all levels of expertise in different forms:

- context for the *interpretation of data.* Context reflects the history of the patient, the diagnosis, or the response to treatment.

- context to account for various *temporal phases* in the course of a patient's disease. Phases may correspond to various *tasks*. For instance, a proposal for managing a sudden deterioration in ventilation during the weaning period - when a patient needs a low level of assistance - could be an adaptation of the therapy given in usual situations.

2) <u>Small variations in expertise among physicians</u>

Although clinical practice is largely based on consensual medical knowledge, each clinical team, specialized in a particular medical field, accumulates specific experience. This difference with commonly accepted knowledge reflects habits, cultural factors, and specific skills. Knowledge representation mechanisms are needed to account for this particular category of variation.

3) <u>Continuous refinement of the expertise of a given team</u>

Medical knowledge is inherently unstable, and varies according to theoretical and experimental research as well as advances in technology. For instance, ventilator management is influenced by improvements in physiological and mechanical models for ventilator-patient interactions, by gradual improvements in medical skills used during care of patients with severe respiratory disorders, by the development of new non-invasive methods that provide real-time information on the patient's status, and by advances in ventilator technology. From a more practical point of view, the design of a knowledge-based system must anticipate continuous modifications of prototypes aimed at adapting their behavior to new real-life situations.

These observations have led us to develop *domain-dependent* software components, which can be reused within an intentionally limited bandwidth.

2.2. Mechanisms for Limited Reuse

The design of intelligent systems requires a blend of software engineering and knowledge engineering technologies and methods (25). We believe that *rule-based object-oriented programming,* mixing rules and objects, is particularly well adapted to our objectives. Indeed, this approach benefits from 1) the abstraction and computation capabilities of object-oriented languages for simulating the relevant entities of an idealized and reified medical world and from 2) the representation capabilities of production rules for modeling the clinician's reasoning. This interest for rule-object alliance is increased by the observation (2) that working expert systems contain only few rule-based modules, the major part being implemented in imperative programming languages.

In this paper, we discuss the powerful symbiosis between 1) two mechanisms of software engineering, i.e., *inheritance* and *encapsulation*; and 2) the ubiquitous mechanism of knowledge engineering, i.e., *forward-chaining rule-based programming.* We identify three main modes of objects / rules combination namely *class inheritance*, *rule-based inheritance*, and *control specification*. We also show, with examples from real-life medical expertise, how these modes contribute effectively to the achievement of limited reuse.

3. Our Epistemological Position

Recent knowledge engineering research has proposed several abstraction paradigms and conceptual architectures to build rationally knowledge-based systems (7,10,52). Despite differences between the structured methodologies that have been developed, a general consensus is apparently emerging about the need to clearly separate three knowledge layers: the domain layer, the inference layer, and the control layer.

3.1. Domain Layer

The domain layer contains all the entities that are of use in representing the application domain. Entities model physical objects in the domain (e.g., patients, doctors, devices, diagnoses, and therapies), as well as intangible objects, concepts, or abstractions that have no physical reality (e.g., therapeutic goals, expected states, and respiratory models). The static structure and dynamic behavior of each entity is defined with the required level of abstraction and detail. A set of relations (physical or conceptual) links entities to each other. For instance, an intensivist (a physician) acts on the ventilator (a device) to modify the respiratory treatment (a therapy) provided to a hyperventilated patient (a diagnosis) and expects the patient to recover (a therapeutic goal) normal ventilation (a respiratory model).

3.2. Inference Layer

The inference layer contains domain inferences which describe what is known about the entities of the domain layer. It represents a discourse about these entities. *Forward-chaining production rules* is an appropriate formalism for representing a large part of medical discourse (44). Some domain inferences are intended to represent specific tasks and must, therefore, be structured into groups. Each group in turn may contain several packs of inferences for each inferential step, such as determination of initial clinical severity, classification of current ventilation, and determination of treatment (action on FiO_2 or action on Peep).

3.3. Control Layer

The control layer (control inferences) contains knowledge about how to use the knowledge of the inference layer. It encodes the strategies used by experts to make a decision. Although eliciting strategic knowledge from experts is a difficult task (23), the importance of explicit formulation of control knowledge and of separation of control knowledge from domain knowledge is now well recognized (8), (11). The main distinction between control and domain knowledge is the nature of the entities they deal with. Control inferences can, therefore, be expressed using the same formalism as for domain inferences. Control inferences are applied to control entities that represent the state of the deductive process at a given time, whereas domain inferences are applied to domain entities. For instance, control inferences enable the user to intervene in the deductive process, to dynamically introduce his/her own strategy, and to modify this strategy as required by the context.

4. Objects and Rules: a Powerful Symbiosis

Object-oriented programming offers a uniform approach to implementing the domain layer. Production rules is a formalism designating model deductive processes. An environment that combines these two approaches is a powerful tool for representing the three knowledge levels described above.

4.1. The Object-Oriented Approach

Object-oriented languages offer a rich set of logical and physical models that can be used to reason about several aspects of modeled systems. Modeling the structures of domain entities and their interconnections using object-orientation provides a layer for simulating the dynamic behavior of the reified medical world. The programming metaphor is based on personifying the physical or conceptual objects from some real-world domain into objects in the program domain (19), (30). Software objects are constructed to represent a monitor, a patient, or a clinician.

Operations on these objects represent domain tasks such as monitoring the respiratory rate, changing the respiratory rate, or administering a new therapy.

Object-oriented programming is a paradigm for developing structured, easily-maintainable software and all the features needed to achieve simulation. *Encapsulation* (or information hiding) supplies a useful barrier between several levels of abstraction. Encapsulation is the complementary concept of abstraction: abstraction focuses on the outside view of an object, and encapsulation prevents the client from accessing the inside view (3). For instance, to modify the level of mechanical assistance, an intensivist can ignore details such as how the pressure support is generated by the servo-valve placed inside the ventilator. At a higher level of abstraction, the organizing principle of *inheritance* allows to describe medical information and create information structures comprising concepts, which are statically related through common property characteristics. Inheritance allows to represent taxonomies of physical or conceptual entities, as well as the generalization/specialization relation (e.g., `Intensivist` inherits from `Clinician`, who in turn inherits from `Physician`). Note that inheritance is not a classification mechanism in the sense of description logics (4). We will see in Section 5.2.2 how we deal with classification in our representation framework.

4.2. Rules and Objects: the NéOpus Architecture

The need for combining object structures and rule-based programming has been widely recognized. The fact base of a rule-based program is a model of the concrete situation that is currently being processed. To bring some semantic structure to facts, one naturally tends to see facts as properties of objects that build up a universe simulating the concrete world. Individual facts are no longer represented as such, and their logical value is ascertained by querying objects in the model. The fact base is thus dissolved in an object-oriented model of the world. This operation is so natural that objects crept into rule-based formalisms (in a rudimentary form) as early as OPS-5.

As a consequence, the main rule-based knowledge representation systems (KEE, ART, and later systems) all have a strong object-oriented component. These object formalisms, however, have usually been defined to suit the reasoning process and rely on frames rather than on objects in the sense of object-oriented programming (e.g., Smalltalk). Accordingly, they are more complex than the standard structures of object-oriented programming languages, i.e., class/instance and inheritance mechanisms. Even when they are defined using such a language, they usually constitute an additional layer on top of the "autochtonous" objects of the language.

We proceeded in the reverse direction, starting with a standard object-oriented language and adding to it a rule-based layer that was "thin" and "seamless" as possible. Our aim was to enrich the class/instance paradigm with rule-based deductive mechanisms. In order to benefit from the work of others we chose Smalltalk-80 as the language for our system, NéOpus (42).

Indeed, forward-chaining rules may be considered a natural extension of ordinary object-oriented programming. An object-oriented program operates a series of updates of a certain set of instances that constitutes a model of the world. Much in the same way, rule-based programming (in its forward-chaining version) uniquely relies on repeatedly updating the fact base (this is no longer true for backward-chaining). The equation "fact base = model of the world" links the two techniques. Their basic difference is only in the control structure, which is rigidly procedural (stack discipline) for object-oriented programming and non-deterministic for rules. Yelland (53) contended that this warranted abandoning rules in his Smalltalk-based knowledge representation system. Based on our experience with NéOpus, we feel on the contrary that this difference does not prevent rules and methods from cooperating smoothly, but rather is precisely the reason why introduction of rules into an object-oriented environment is worthwhile.

5. Three Modes of Knowledge Reuse

We propose three main categories of limited reuse, which are represented using three embedded mechanisms addressing three different *levels* of granularity: 1) class inheritance, 2) combination inheritance/rules, and 3) rule base inheritance. We will illustrate these three mechanisms with various examples extracted from medical expertise for mechanical ventilation management. To facilitate understanding of our examples, we will first describe the medical problem.

5.1. The medical problem

Mechanical assistance provided to a patient with respiratory insufficiency must be well adapted to his or her physiological needs: an excessive level of assistance results in unacceptable *hyperventilation* and an inadequate level imposes extra work on the patient's respiratory muscles. The clinician must assess the *respiratory comfort* of the patient and the time-course of his or her ventilation and must select the ventilator settings accordingly. A second task of the clinician is to reduce mechanical assistance gradually until the patient is able to breathe alone. This is known as the *weaning process* . This process can be complex, especially in patients who have been on prolonged mechanical ventilation. In such cases, the clinician relies on the considerable expertise that he or she acquired during long clinical practice. The patient's ability to breathe alone must be assessed before disconnection from the ventilator, in order to avoid repeated connections and disconnections.

To facilitate and improve the weaning process, we designed the knowledge-based system called NéoGanesh. It is based on the knowledge of weaning management acquired by the clinical staff of the Intensive Care Unit at the Henri Mondor Hospital. In contrast to other systems (26,48), our system deals with a voluntarily limited problem: 1) only one mode of ventilation is managed by the system, i.e., pressure support ventilation (PSV); 2) patients ventilated with the system must have spontaneous respiratory activity. These limitations allowed us to design a *closed-loop system* that controls the ventilator without any intervention by the clinician.

5.2. Class Inheritance to Refine the Expertise

We will describe in detail how class inheritance allows to take into account small variations in several specific situations encountered in mechanical ventilation management.

5.2.1. Small Variations According to the Context of Application

When interpreting data, it is essential to consider the context in which the data were collected. In an analysis of the basic nature of the diagnostic process, Rasmussen (46) pointed out that diagnostic judgment does not consist in theoretical categorization of observed data but rather in a search for several alternatives, which can be separated depending on the actual context of decision making. In medical reasoning, the notion of context is central for interpreting physiological parameters.

One of the first systems that introduced explicit management of context is VM (20), a precursor system designed to interpret on-line physiological data in intensive care units. In VM, physiological data are represented symbolically, with each symbol representing a particular range of values (such as "acceptable" or "low"). As pointed out by Fagan, the meaning of a given symbol varies as the patient moves from state to state. To specify symbolic interpretations for each parameter in the context of a particular therapeutic state, VM uses *initializing rules* that are invoked when a state transition is detected. These rules assign symbols to the ranges corresponding to the new context. For example, the rule "initialize-CMV" sets the "acceptable" range for "mean arterial pressure" to the interval [88 torr - 95 torr] as soon as the patient is placed on controlled mechanical ventilation (CMV). The problem with this architecture is that the various sets of ranges are defined independently from each other. When a new set of ranges has to be defined, the ranges of *all* the symbols must be systematically specified.

Class inheritance can substantially reduce the quantity of information needed, by factoring out the ranges that do not change. In our context, the sets of ranges correspond to various expectations of the internist regarding data measurement results. Representing expectations by classes (`RespiratoryExpectation`) allows to simplify the representation and assignment of symbolic ranges. Each `RespiratoryExpectation` is represented by a class, which redefines only the range for the context-dependent parameters (see Figure 1). The other ranges are inherited. When a transition to a new mode is detected, the clinician changes his / her expectations to classify the current ventilation of the patient.

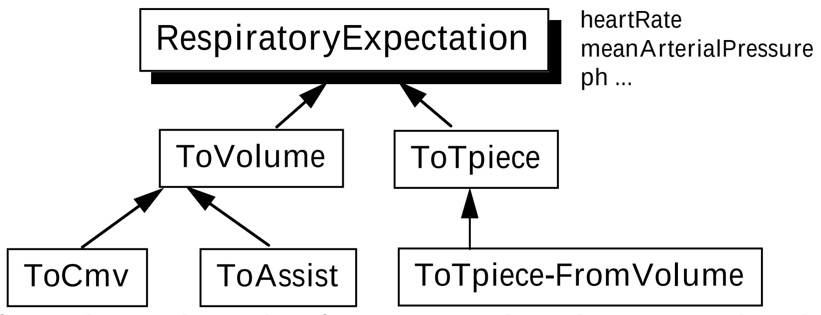


Figure 1 : The figure shows a hierarchy of expectations about the patient's physiological parameters when the ventilator is switched to a new mode, i.e., to Controlled Mandatory Ventilation (ToCMV), Volume Controlled Ventilation (ToVolume), or Assisted Ventilation (ToAssist), when the patient is placed directly on a T-piece (ToTpiece), or when the patient is placed on a T-piece after a period of volume-controlled ventilation (ToTpiece-FromVolume) (adapted from (26)).

5.2.2. Representing the Taxonomy of Possible Solutions - Part 1

Well-structured diagnostic problems can be solved by the method of heuristic classification. This method involves three steps: 1) abstraction from input data; 2) association with a taxonomy of possible solutions; and 3) refinement of the collection of potential solutions (10). Embedding classification mechanisms into object-oriented languages is not, however, a trivial task. Yelland (53) for instance, proposed a scheme for integration of subsumption mechanisms smoothly integrated with object structures. The main problem with these powerful mechanisms is that they are not complete (for ex., see (27) for an evaluation of various terminological systems). In our context, we believe that the sophistication of subsumption mechanisms is not needed, since the collection of potential solutions is to support only *limited extensions*. Instead, we propose use of class inheritance to represent the taxonomy of possible solutions (see Figure 2).

In NéoGanesh, each class is characterized by a set of constraints, to which patient's data is matched for classification. For instance, to diagnose the current respiratory state of a patient, clinicians use three main parameters: respiratory rate (RR), volume inspired at each breath (Vt), and pressure of carbon dioxide at the end of expiration ($PetCO_2$). Consequently, the class `Normal`, which represents normal ventilation, is characterized by the following set of constraints (expressed as a Boolean expression):

```
Constraint for class "Normal"
        Vt > 250 AND
        (RR < 28 AND RR > 12) AND
        PetCO2 < 55
```

The classification process consists in comparing a given ventilation state (represented by an instance of class `MeasuredVentilation`) in a patient to the solution classes. Instead of systematically matching against all the classes in the taxonomy, solution classes are grouped into several collections, each of which represents a particular combination of available parameters. For instance, the solution classes that require parameters Vt, RR, and $PetCO_2$ are: `Normal`, `Tachypnea`, `Bradypnea`, `SevereTachypnea`, and so forth. This collection of classes is represented as a method (called `standardSolutionList`) defined in the class that represents the current clinician (`Intensivist`). Other collections include `fineSolutionsList`, `rawSolutionsList` and so forth.

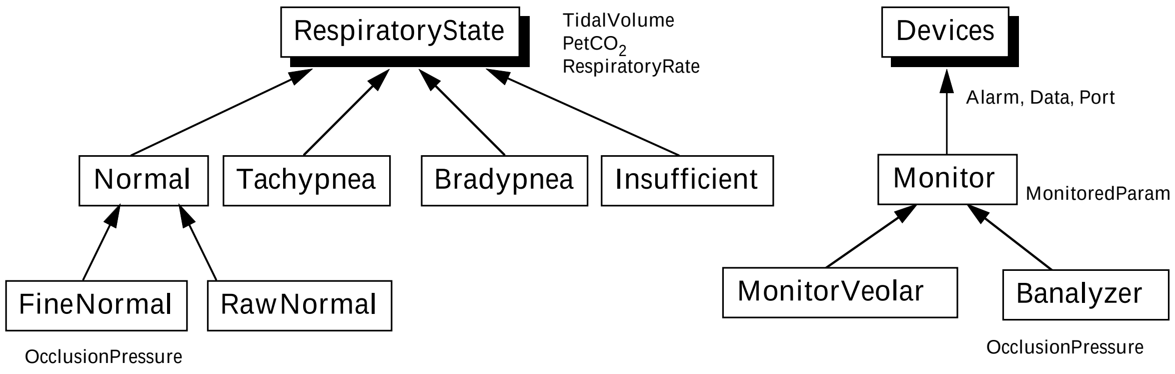


Figure 2. The left-hand panel shows a part of the set of solution classes used by the intensivist to classify the patient's ventilation. This taxonomy is refined when a new combination of available parameters is introduced. Several classes of monitors are defined (right-hand panel). Each class of monitor measures different physiological parameters. The class of the monitor (among other things) determines the list of the solution classes that will be used to classify the patient's respiratory status.

5.2.3. Representing the Taxonomy of Possible Solutions - Part 2

The definition attached to each element of the taxonomy of solutions, as well as the thresholds used for interpreting physiological data, may vary according to the patient's characteristics (sex, age, and diagnosis), the clinical context, or the physician. Several short examples extracted from different areas of medical expertise will show how class inheritance and encapsulation allow to introduce such variations.

Refinement of the potential solutions may depend on the availability of certain data. To characterize the patient's respiratory state with greater precision, we can *add* a measure called occlusion pressure (P0.1), which reflects the effort that the patient must supply to trigger mechanical assistance. To represent this new piece of information, we added to the NéoGanesh taxonomy new solution classes with constraints that explicitly refer to this new data. For instance, the additional solution class `FineNormal` is a subclass of `Normal` with the following constraints:

```
Constraint for class "FineNormal"
          VT > 250 AND
          (RR < 28 AND RR > 12) AND
          PetCO2 < 55 AND
          P0.1 > -3
```

Similarly, we individualized `Tachypnea` subclass `FineTachypnea`, characterized by a constraint with an additional condition on P0.1. This new list of solution classes is represented as a new method (`fineSolutionsList`) in class `Intensivist`.

This scheme for refining the solution class taxonomy can also be used to represent a *lack* of information. For instance, a bad signal from the gas analyzer does not allow to obtain reliable $PetCO_2$ data. To account for this lack of information, we introduce a new collection of solution classes in our taxonomy (`rawSolutionsList`). This list include class `RawNormal` (another subclass of `Normal`) whose constraint does not include the condition on $PetCO_2$; class `RawInsufficient` and so forth (see Figure 2).

Any combination of available data is represented by a list of classes, each of which represents a particular constraint on the values of available data. Whereas the number of classes increases exponentially in theory, in practice only classes representing relevant sets of available data are represented. In NéoGanesh, the following classes have been introduced to date:

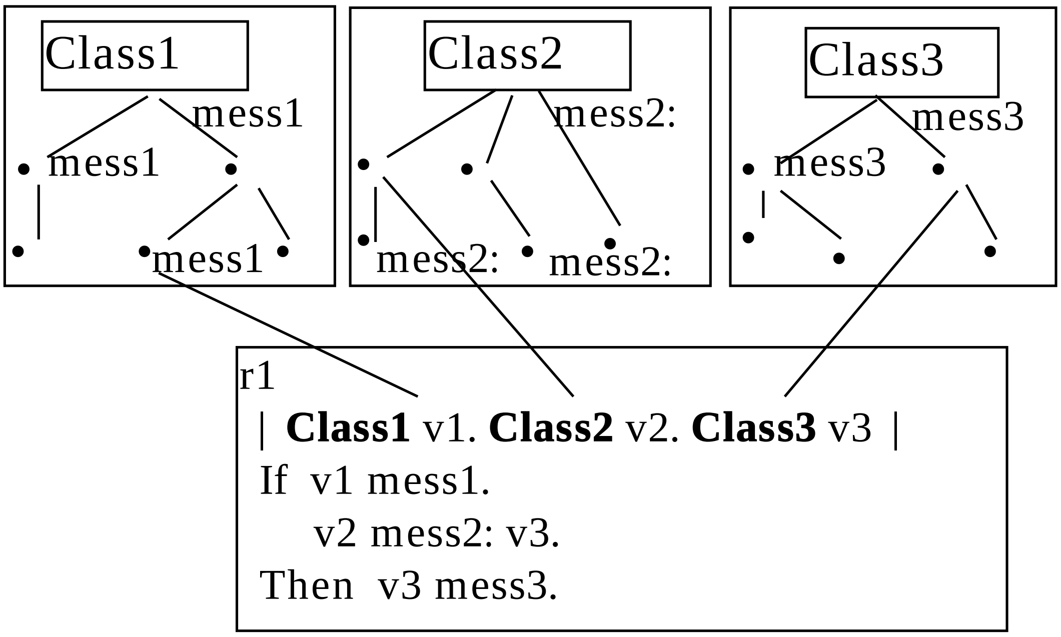
- normalSolutionList = Normal, Tachypnea, SevereTachypnea, Bradypnea, Insufficient, HyperVentilation;
- rawSolutionsList = RawNormal, Tachypnea, SevereTachypnea, RawInsufficient, HyperVentilation;
- fineSolutionsList = FineNormal, Tachypnea, SevereTachypnea, FineInsufficient, HyperVentilation, Bradypnea.

Note that all combinations of classes are not necessarily present. For instance, the list `rawSolutionsList` does not contain hypothetical classes `RawTachypnea` or `RawSevereTachypnea`, which do not make sense since parameter $PetCO_2$ is not used in the constraints attached to the corresponding classes (`Tachypnea` and `SevereTachypnea`).

The choice of the list of solution classes to be used for classification is, therefore, determined by the set of available parameters, which is in turn directly provided by the class of the monitor used.

5.3. Combining Inheritance with Rules

The preceding examples show how inheritance can be used to factor information. Class inheritance can also be used *in conjunction with* rules, yielding another dimension of limited reuse. This is achieved in NéOpus by so-called "natural typing" of rule variables. The idea behind natural typing is to allow the pattern-matcher to consider direct instances as well as instances of subclasses to be matched by rule variables for a given rule. The interpretation of a rule is, therefore, dynamic, since 1) the condition and action parts of the rules are entirely expressed in terms of messages sent to the matched objects; and 2) the messages are redefined in subclasses. In other words, the rules are *context-dependent* where the context is represented by the set of objects that match the rule. More precisely, let r be a rule that declares n variables v_i (i=1, n). and C_i be the class declared for v_i. If each C_i has k_i concrete subclasses, the total number of possibly different interpretations of r is Πk_i (i = 1, n). In practice, if r has 3 variables and each declared class has 5 concrete subclasses, then r has $5^3 = 125$ possibly different interpretations! The figure 3 illustrates the notion of natural typing.



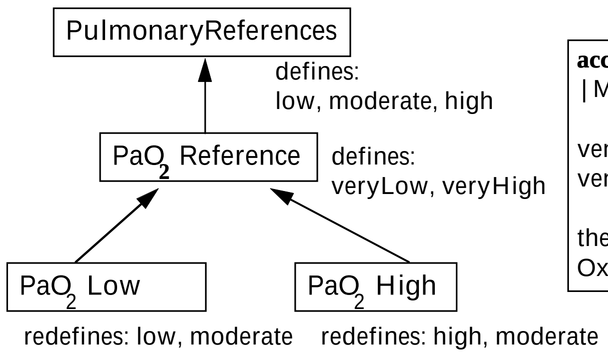
```
r1
| Class1 v1. Class2 v2. Class3 v3 |
If  v1 mess1.
    v2 mess2: v3.
Then  v3 mess3.
```

Figure 3. A rule with the corresponding class hierarchies. The rule should be read as follows: For any v1, instance of class Class1, v2 instance of Class2 and v3 instance of Class3, if "`v1 mess1`" and "`v2 mess: v3`" evaluate to true, then evaluate "`v3 mess3`". The rule uses three messages (mess1 through mess3) that are defined and redefined in the classes of the corresponding hierarchies.

5.3.1. Example of the Use of Natural Typing

At LDS Hospital (Salt Lake City, USA), East et al. (18) developed computerized protocols to assist clinicians with the difficult task of ventilating mechanically patients with *Acute Respiratory Distress Syndrome* (ARDS). They evaluated the arterial oxygenation of the patient by means of PaO_2 measurement to determine the best therapy approach. If barotrauma is diagnosed, lower arterial pressure of oxygen (PaO_2) are tolerated and steps are taken to reduce alveolar pressures. This tolerance allows to shorten therapy.

To introduce new thresholds for tolerance, we reify the references of PaO_2 (see Figure 4, left) to allow their modification by subclassing. Specific objects (`Strategy`) may be used to specify, in the rules, all aspects of the intensivist's attitude (definition for oxygenation classification, size of therapy change and frequency). The intensivist adopts the relevant strategy depending of the examination of supine chest radiographs (for instance detection of barotrauma). The chosen strategy defines the values of reference (`PaO2Reference`) that should be used for classifying PaO_2. Then the rules, which are used for arterial oxygenation classification, match all subclasses of `PaO2Reference` (see example on Figure 4, right). Thanks to inheritance and encapsulation, *conditions part of rules* are differently interpreted depending on the nature of the matched object representing a PaO_2 reference.



```
acceptableOxygenation
 | MeasuredVentilation vent. PaO2Reference reference |

vent paO2 > reference low.
vent paO2 < reference moderate.

then
Oxygenation new acceptable.
```

Figure 4: Left: gradual introduction of modification of thresholds for arterial pressure of oxygen classification. Right: an example of rule to classify arterial oxygenation. Methods `low` and `moderate`, which return corresponding threshold values, are redefined in subclasses of `PaO2Reference`.

5.3.2. More on Thresholds

The thresholds used by the intensivist to classify the ventilation are dependent on patient's characteristics, especially the physiological disturbances. Figure 5 shows a hierarchy of disorders. Figure 6 shows a rule that uses this hierarchy in combination with the hierarchy of intensivist classes (see Figure 7). The rule indicates that higher respiratory rates should be tolerated in patients with conditions that imply neurologic alterations (Cf. method `frequencyToleranceFor`). This method modifies the ventilation models accordingly. Again, inheritance and encapsulation are used to interpret the rule depending on the context: here, the context is defined by the set of objects matched by the rule, i.e., the intensivist, the patient, and the disorder.

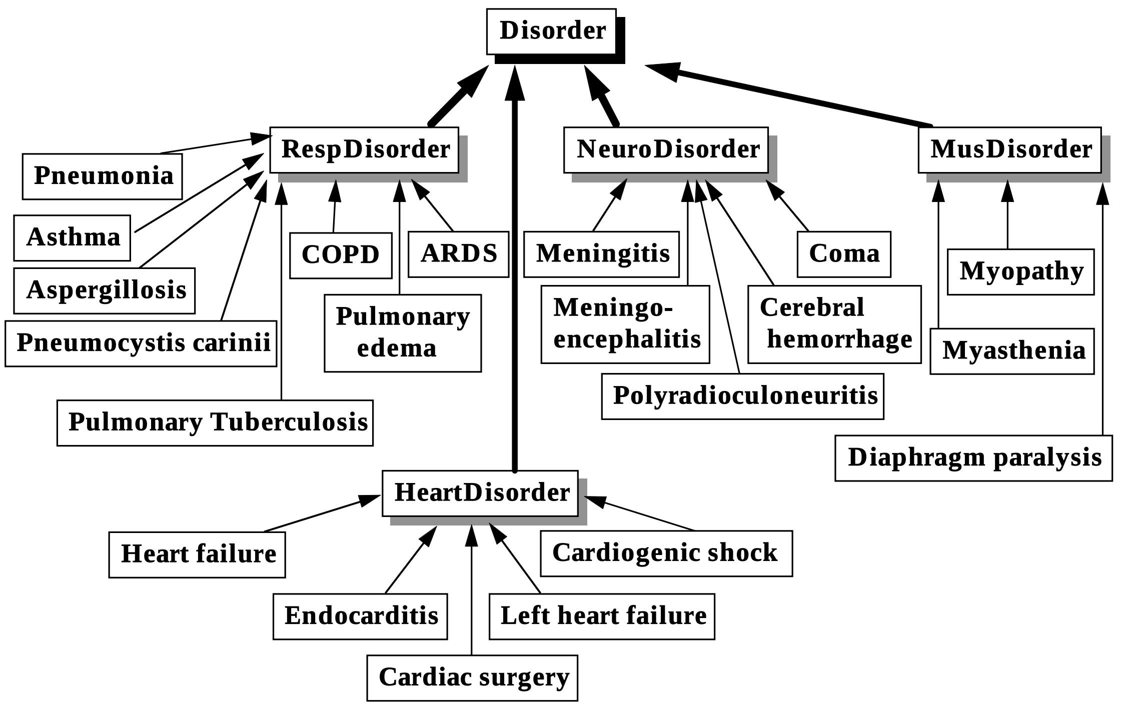


Figure 5: An example of taxonomy of some disorders in intensive care unit patients. Each class of the inheritance tree redefines method `impliesNeurologicAlterations`, used in rule of Figure 6.

5.3.3. Variations among Physicians

Here is a third typical example of the use of natural typing in the NéoGanesh knowledge base to represent variations across physicians.

```
tolerateHighRRForNeuroDisorders
   |Intensivist expert. Patient patient. Disorder disorder|
  expert takesCareFor: patient.
  patient hasDisorder: disorder.
  disorder impliesNeurologicAlterations.
then
  expert frequencyToleranceFor: disorder.
```

Figure 6: An example of rule that adapts interpretation for Respiratory Rate (RR) measurement in the context of a neurologic disorder. Note also that method `frequencyToleranceFor:` is redefined in subclasses of class `Intensivist`.

Various strategies can be used to adapt the rate of assistance reduction to the improvement in the patient's condition. In VIE-VENT (33), an open-loop system for the control of ventilation in paediatrics, three different strategies for withdrawing mechanical assistance (called *weaning* strategies) are available, depending on the psychological characteristics of the physician: conservative, normal, or aggressive. In each strategy, parameters to be considered, tolerance of variations, and intervals between invasive blood gas determinations are different.

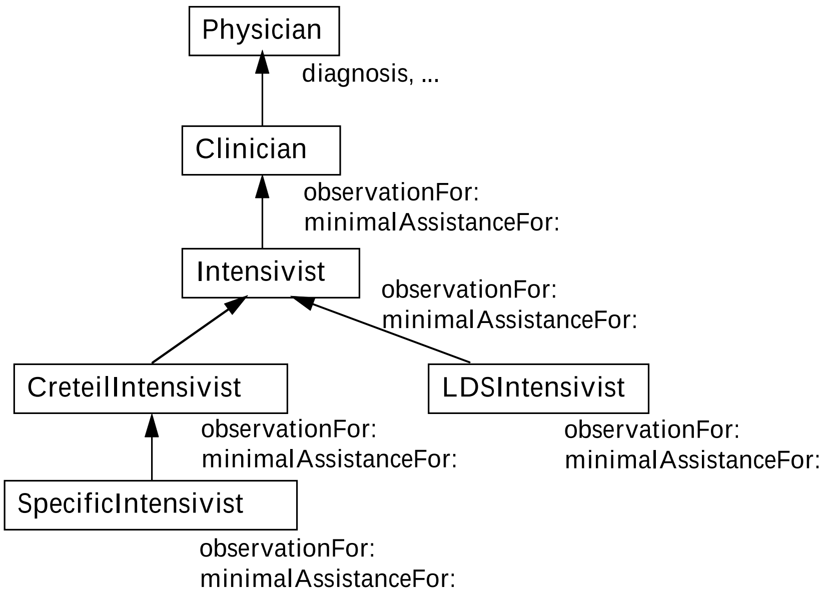


Figure 7: The hierarchy of physicians.

NéoGanesh offers two weaning strategies. A patient must tolerate minimal assistance during a certain period of time to be declared weanable. In the first strategy (*the standard strategy*), the duration of the observation period is fixed automatically by checking the level of assistance required by the patient after one hour of ventilation: if the level is less than 15 cmH_2O the period is 2 hours, or the period is 12 hours if the level is higher than 15 cmH_2O. The second strategy is "open", i.e. the clinician in charge of the patient can customize the strategy to his / her preferences regarding duration, level of assistance, and tolerance of instabilities.

```
endOfObservation
|Intensivist expert. Patient patient. Ventilator ventilator.
Local state|
ventilator connectedTo: patient.
expert takesCareOf: patient.
state := expert expectedStatefor: patient.
state isPersistent.
state class = PreWeanable.
state duration > expert observationFor: patient.
ventilator   levelOfAssistance   =   expert   minimalAssistanceFor:
patient.
then
expert expectedFor: patient state: Weanable new.
```

Figure 8: An example of rule for the weaning strategy. The methods `observationFor:` and `minimalAssistanceFor:` are redefined in subclasses of `Intensivist`.

These two strategies are represented by two different subclasses of class `Intensivist`: `CreteilIntensivist` for the standard strategy and `SpecificIntensivist` for the open strategy (see Figure 7). These two subclasses redefine the methods understood by intensivist objects that appear in the condition parts of observation strategy rules (see Figure 8 for an example). Note that the same design could be used to represent the three strategies of VIE-VENT.

5.3.4. Another Example of Natural Typing: Management of Time

In most medical knowledge-based systems, the course of the disease is divided into several phases. These phases may correspond to *steps* in the treatment (e.g., in cholesterol-lowering treatment (47) or respiratory assistance (20)) or to various *states* of time-course of the patient's condition (such as hemodynamic states (12), (32)). In each phase, specific rules are applied.

Recently, we have proposed a model for temporal reasoning representation in real-time systems (17). This model is based on temporal abstractions that allow observations to be interpreted incrementally as they are acquired. Two mechanisms are used: *aggregation* of similar situations and *forgetting* non-relevant, redundant, or out of date information. Activation of these two mechanisms is context-dependent.

Context change and time management are taken into account using our combination of class inheritance and rules. This is achieved by 1) reifying *phases* to build a hierarchy of classes representing various significant phases in the course of the disease and, 2) introducing an explicit management of time that accounts for state changes and time-courses.

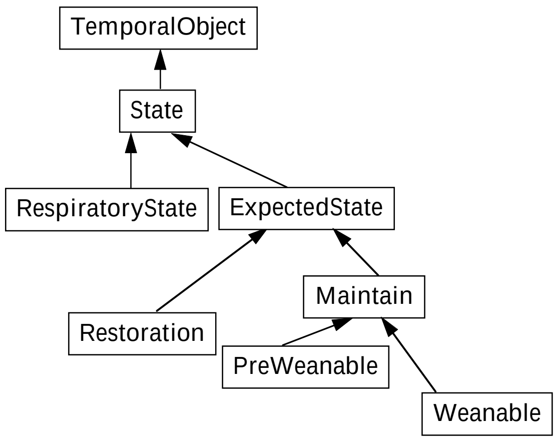


Figure 9: Part of the hierarchy of temporal objects in NéoGanesh.

Temporal objects are time-stamped entities, used to develop a temporal discourse about the time-course of the patient's ventilation. One of the goals of medical reasoning is to assess the stability of ventilation over periods of time, the duration of instabilities, and the persistence of inadequate therapies. Thus, the clinician seeks to perceive significant changes between successive observed states or successive expected states. This perception of changes is, of course, context-dependent: for instance, the notions of *similarity* and *dissimilarity* between states depends on the class of the states compared. This is, once again, represented by methods attached to the corresponding subclasses of `TemporalObject`. Figure 9 shows parts of the hierarchy of temporal objects used in NéoGanesh. `RespiratoryState` represents the ventilation status of the patient. `ExpectedState` represents the future patient respiratory state anticipated by the physician. Figure 10 shows a rule taken from the rule base that handles perception of change.

```
partialContinuity
|State s1 s2 s3. Duration d1|
 s3 persistent.
 s1 similarTo: s3.
 s2 dissimilarTo: s3.
 s2 between: s1 and: s3.
 s2 duration <= d1.
then
 s1 validDuring: s3 duration.
```

Figure 10: A rule that aggregates two disjoint states (`s1` and `s3`), separated by an instability (`s2`) considered short enough to be discarded. This rule matches all the instances of `State` and of its subclasses.

5.4. Representing Control

Decision-making models are usually structured as a sequence of tasks. Oxygenation protocols for controlled positive pressure ventilation (18) require a sequence of tasks, e.g., data acquisition, oxygenation classification protocol, core protocol, and therapy instruction for the user. Similarly, the treatment of anemic patients is organized into several steps. In Therapy Advisor (44), a system applied to this complex therapeutic problem, each inferential step of the model of therapy planning is represented by a frame, which i placed inside an agenda, and metarules manage the execution of each inferential step. NéoGanesh contains a complex sequencing function for reasoning (see details in the section 6.2.3). However, the normative sequence can be broken by heuristics in specific situations. Thus, availability of an explicit representation of the sequence is important to allow modification of the sequence according to the context.

```
badTherapy
 |ClinicEvaluator evaluator.|
 evaluator status =#loop.
 evaluator currentStrategy = #standard.
 evaluator case class = Restoration.
 evaluator  case  durationInExpertise  >  e  case  diagostician
toleranceInstabilities.

then
 evaluator case diagnostician changesTherapyTo: #specific
```

```
pbWithClassification
 |ClinicEvaluator evaluator |
 evaluator status = #loop.
 evaluator currentStrategy = #specific.
 evaluator case class = Restoration.
 evaluator  case  durationInExpertise  >  e  case  diagostician
toleranceInstabilities.
then
 evaluator case diagnostician checkModelsForClassification.
```

Figure 11: The metarule `badTherapy` states that when a expected state (restoration of a normal ventilation) is not reached despite some actions, the clinician in charge has to change his / her therapy. The metarule `pbWithClassification` indicates that if the problem is persistent in spite of a modification of the therapy, the clinician must verify the model used to classify the ventilation in order to adapt it if needed the observed data.

We use objects called *Evaluators* to reify the state of the current reasoning process at a given time and production rules (called *metarules*) to represent the control inference. In reifying the control, several aspects of the control can be explicitly specified. Metarules match evaluator objects to intervene on the deductive process.

Stop-rules controlling termination of a search are not usually formulated explicitly (46). When evaluator objects are used, this is no longer the case.

Evaluators own a specific stop condition. Thus, a metarule can stop the reasoning process when a stop condition is satisfied. In the ventilation classification, a stop condition, which is true when the input data match a respiratory model, is attached to an evaluator. Classification is stopped when the evaluator condition is true as indicated by a metarule (see Figure 11). By subclassing class `Evaluator`, specific control objects can be introduced. For instance, `EvaluatorAgenda` contains a list of packs of rules that represents the sequencing of inferential steps. Meta-rules are used to recognize whether or not a step has been performed, the outcome, and which step must be taken next.

An expert switches between different strategies by transitions that are cued by observed data. At the meta level, we describe the change of strategy. Thus, the strategies change dynamically during the decision process (see Figure 11).

5.5. Rule Base Inheritance

We introduce a third level of representation of limited reuse: Rule Base Inheritance (RBI). RBI allows to gradually specify the context of application of rules by matching objects that are relevant for this context.

5.5.1. The Mechanism

In our representation framework (NéOpus), rules are grouped in *rule bases*, which are represented as abstract classes. The idea of transposing the class inheritance mechanism to rule bases flows naturally from this. In an earlier paper (40), we described a scheme for transposing the *intuition* of class inheritance to rule bases. In this scheme, each rule base can be defined as a sub-base of an existing rule base, and therefore inherits all its rules. An overriding mechanism, based on rule names, allows a rule base to *redefine* an inherited rule into a more specific rule. This scheme is based on static propagation of rule compilation to sub-bases, together with a particular *control strategy* (RBI strategy) aimed at ensuring that, in case of conflict, rules defined in the lowest sub-base will be selected.

There are indeed several advantages in providing rule bases with an inheritance mechanism. This approach provides a high level scheme for organizing rules, allows to factor out common rules, and simplifies control strategy specification. We will show here how RBI can be used to provide explicit contextual dependency of medical knowledge.

5.5.2. RBI for Context Dependency in Temporal Reasoning

Figure 12 shows an example of a rule base inheritance tree in NéoGanesh. In this example, inheritance is used to gradually introduce context-dependency in temporal reasoning.

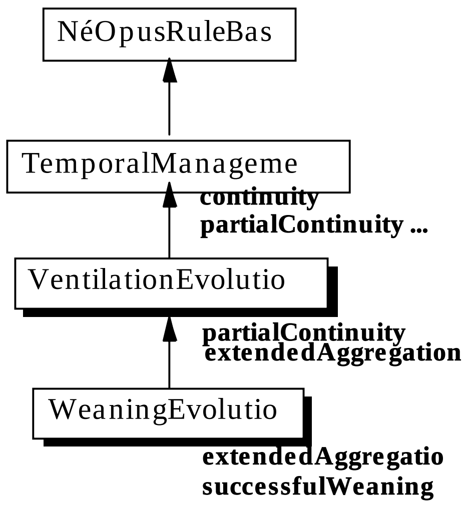


Figure 12: An inheritance tree of rule bases.

At the highest level, rule base `TemporalManagement` contains rules that define the perception of time (continuity and discontinuity between successive events) in a use-neutral manner. These rules match general temporal objects (instances of class State). Rule base `VentilationEvolution` is then introduced to refine this general-purpose temporal reasoning in a way that is specific to ventilation management. `VentilationEvolution` is defined as a sub-base of `TemporalManagement`, and therefore inherits all the general-purpose rules, via the RBI mechanism. Moreover, `VentilationEvolution` redefines some inherited rules (such as `partialContinuity`) and adds specific rules (such as `extendedAggregation`). Rules defined or redefined in `VentilationEvolution` match more specific objects (instances of class `RespiratoryState` instead of the general `State` class, see Figure 13).

```
partialContinuity (redefined in VentilationEvolution)
   |RespiratoryState s1 s2 s3|
 s3 persistent.
 s1 similarTo: s3.
 s2 dissimilarTo: s3.
 s2 between: s1 and: s3.
 s2 durationInExpertise <= 1.
 s3 durationInExpertise > 1.
then
   s1 validDuring: s3 duration.
```

Figure 13: Rule `partialContinuity` is redefined in a sub-base of `TemporalManagement`. The rule matches specialized objects (`RespiratoryState` instead of `State` in the first version of the rule) and redefines the condition part. The rule indicates that two disjoint respiratory states, separated by an instability that lasts less than 1 expertise cycle, can be aggregated.

Similarly, the rule base `WeaningEvolution` applies to situations (modelled by `Weanable` class, a subclass of `RespiratoryState`) in which the patient is on the verge of breathing without assistance, and is therefore a sub-base of `VentilationEvolution`. The rule `extendedAggregation` (inherited from `VentilationEvolution`) is redefined to tolerate moderate ventilation instabilities. Rule `successfulWeaning` indicates that the patient can be extubated. At this level, rules match only instances of class `Weanable`.

5.5.3. Example: Variations in the Therapy (planification of actions)

RBI is also used to represent different therapeutic strategies depending on the context. In NéoGanesh, rule base `TherapeuticActionRules` (see Figure 14) represents the standard therapy. It groups the rules that define the specific actions to be performed on the ventilator at the end of the reasoning. Rules `ActTachypnea` and `ActInsufficient` increase the level of assistance in the event of tachypnea or insufficient ventilation, respectively; rule `ActHyperventilation` decreases assistance in the event of hyperventilation; and so on. Rule `ActNormal` maintains the current level of assistance if the patient's ventilation is normal.

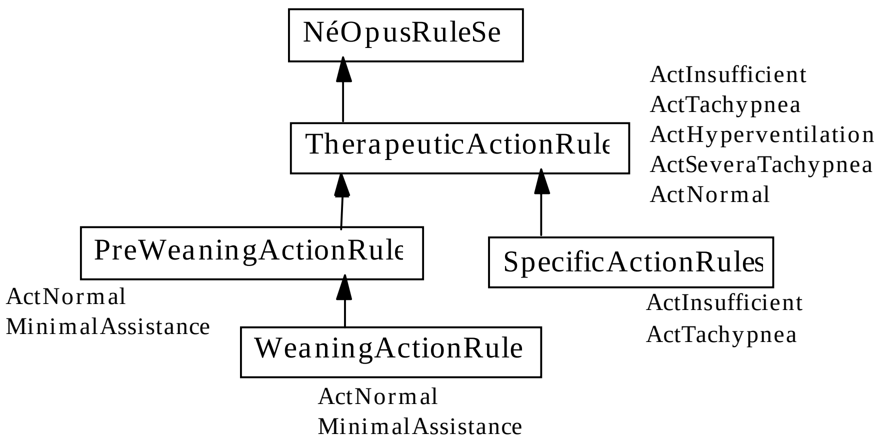


Figure 14: An inheritance tree of rule bases for therapeutic actions.

However, some strategies must redefine some actions on the ventilator. For instance, the *Preweaning* strategy is used to test the ability of the patient to breathe without external assistance. When Preweaning is active, the level of assistance must be set to the minimal value, if the patient's ventilation is normal. This peculiarity is represented simply by using RBI. `PreweaningActionRules` is defined as a sub-base of `TherapeuticActions` (see Figure 14), and therefore inherits all rules that represent the standard therapy. The rule base redefines rule `ActNormal` (see Figure 15).

Similarly, rule base `WeaningActionRules` contains knowledge to be used when ventilation deteriorates suddenly after the patient was declared weanable. `SpecificActions` represents a strategy to accentuate assistance modifications in specific situations. These variations are also represented using RBI and rule redefinition.

```
actNormal (TherapeuticActionRules)
|Intensivist expert. Patient patient. Ventilator ventilator|
ventilator connectedTo: patient.
expert takesCareOf: patient.
patient ventilation isNormal.
then
expert doeNotChangeAssistanceOn: ventilator.
```

```
actNormal (PreweaningActionRules)
|Intensivist expert. Patient patient. Ventilator ventilator|
ventilator connectedTo: patient.
expert takesCareOf: patient.
patient ventilation isNormal.
ventilator levelOfAssistance > expert minimalAssistanceFor:patient
then
expert placesMinimumAssistanceFor: patient on: ventilator.
```

Figure 15: Rule `ActNormal` inherits from `TherapeuticActionRules` is redefined in `PreweaningActionRules` rule base.

6. Application of our Methodology: the NéoGanesh System

To demonstrate that our methodology is practical, we will supply additional concrete details on the NéoGanesh system in this section.

6.1. The NéoGanesh System

NéoGanesh continuously monitors ventilation, respiratory rate (RR), tidal volume (Vt), and end-tidal CO_2 pressure ($PetCO_2$). The level of pressure support (PS) provided to the patient is used to evaluate the quality of the respiratory system. All these parameters are obtained from external devices via serial communication

(Rr, Vt, PS are obtained from a ventilator and $PetCO_2$, from a CO_2 analyzer). The system modifies the ventilator settings (PS and ventilatory mode), also via serial communication. It is connected to a ventilator Veolar [Hamilton Switzerland] and a CO_2 analyzer (Novametrics 1250, US). NéoGanesh works on a PC compatible using Smalltalk-80 (under Windows 3.1 environment) with NéOpus. The interface was designed using the Model View Controller paradigm (22). By acting on the mouse, the user feeds information about the patient into the system and launches a ventilation expert consultation. A simulation mode is also available to enable the user to interact with the system and to modify physiological parameters, in order to test typical situations.

The average duration of one cycle of reasoning is about 1 second on a PC/486, and 2 minutes are required for data acquisition (sample frequency 0.1 Hz). A typical session with a patient controlled by the system involves about 350 reasoning cycles (one day).

6.1.1. Clinical Results

NéoGanesh is a prototype in use at the Henri Mondor Hospital. NéoGanesh has ventilated more than sixty patients with promising clinical results. A first clinical study (19 patients) showed that the system was able to adapt assistance to patients' needs while gradually lowering the level of assistance, thereby facilitating the weaning process (14). In second study (38 patients), NéoGanesh improved the prediction of weaning outcomes, as compared with conventional practice (16). These results incited us to undertake our current clinical studies in larger populations of patients to confirm the validity of our approach.

6.2. Knowledge Representation

In conformity with our epistemological position (see §3), our system design is based on three knowledge layers: the **domain layer,** which includes objects defined by Smalltalk classes; the **inference layer,** represented by forward-chaining rules; and the **control layer** represented by metarules used to control the firing of rules.

6.2.1. The domain objects

The system includes a representation of every real object involved in the ventilation process. We defined 81 classes and about 350 methods to describe the medical problem. We will now describe the most important class, i.e., the `Intensivist` class, a subclass of the `Clinician` class.

The intensivist can modify the settings of the ventilator. His/her knowledge is represented by production rules, as explained in the next section. The intensivist object responds to messages that evaluate the patient's status, such as `currentDiagnosis`, which represents the state of ventilation (an instance of class `Normal`, `Tachypnea`, ...), and `severity`, which depends on the level of assistance needed by the patient. The intensivist object also manages various thresholds needed to qualify correct ventilation (e.g., `frequencyReference` that returns an interval with the lower and upper limits for the considered patient, or `observationFor` that defines the duration of observation before decreasing assistance; see Figure 8). These messages are used in the conditions parts of the rules that implement the intensivist's knowledge.

6.2.2. The Rules

In our system, the knowledge of the intensivist is represented by NéOpus rules, operating in forward-chaining and grouped in rule bases. The system contains eleven rule bases. Seven rule bases, containing a total of twenty-one rules, are dedicated to the diagnosis of current ventilation and to the definition of therapy. For these rules, conditions parts typically test the values of the physiological data, information about the patient, or the time-course of the patient's ventilation (see Figures 6, 8, and 15), using the messages defined in the corresponding classes. The rule actions set the intensivist object in motion, leading to side-effects, such as modifications of the overall check-up and adaptation of assistance (see Figure 14). Rule bases dealing with planification of actions are organized in an RBI hierarchy (Cf. Section 5.5.3). In this hierarchy, five rules are inherited and seven are redefined in sub-bases.
Four rule bases are dedicated to the representation of temporal reasoning. These rule bases are organized in an RBI hierarchy (Cf. Section 5.5.2). They contain a total of twenty rules (four are inherited in sub-bases and six are redefined in sub-bases).

6.2.3. The Control

In NéoGanesh, the control of reasoning is a complex task. The reasoning has a sequential structure and several tasks must be ordered. However, in alarming situations (bradypnea, persistent apnea, and patient disconnection) the sequencing must be broken and a set of heuristic is used to short-cut several inferential steps. The control strategies were introduced explicitly using the declarative representation of control and RBI for meta bases. A total of nine meta bases were defined. Figure 16 shows a part of the inheritance tree of rule bases and meta bases. The metarules at the lowest level (MetaClinicAlarms) are triggered preferentially to rules from higher levels. They represent control metarules for alarming situations. Meta base MetaClinicProtocols contains metarules that deal with the sequencing of the reasoning. Note that the three first levels are independent of the application.

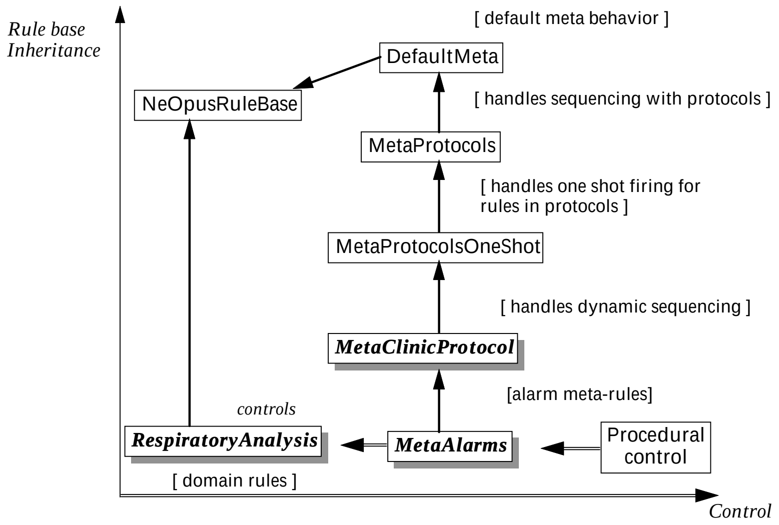


Figure 16: This figure shows an excerpt of the hierarchy of meta bases (at right) and the hierarchy of rule bases (at left) in NéoGanesh. Bold rectangles indicate rule bases specific to the domain. Other rule bases and meta bases are generic.

To control reasoning for diagnosis and definition of therapy, eighteen metarules are defined (five are inherited and nine are redefined in sub-bases). For control of temporal reasoning nineteen metarules are defined (five are inherited and eleven are redefined in sub-bases).

7. Discussion

We will discuss the disadvantages and advantages of our methodology as compared with other approaches.

7.1. Problems with Declarative Knowledge Representation

Our approach directed by the classical object-oriented perspective (so-called message passing) has been challenged by knowledge representation specialists (for ex., see (38,43)). Objects are viewed as closed entities that can be addressed only via the interface defined by their class. In particular, obtaining the logical value of their properties requires the use of procedures that must be explicitly defined in their class, whereas in a frame-based model one has direct access to the slots, which are felt to carry semantic information.

Indeed, the encapsulation principle gets in the way by hiding the object structure (looked upon as mere implementation detail by software engineers) and hampers the declarative and explicative capacities of the system. This problem is visible in at least two places in the NéOpus system and requires additional information from the programmer: 1) it creates a need for declaring explicitly which objects have significantly changed their state as a consequence of rule firing (the `modified` action) (41) and 2) it makes it impossible to elicit the goal of a rule from its text, creating a need for an additional field in the rule format to provide this type of information (which we have called an *assertion* in (41)). These characteristics are clearly at variance with the principles of declarative knowledge representation. However, we feel that the benefit gained from potentially applying rules to the whole universe of object-oriented models created by object-oriented programmers justifies these departures from generally accepted principles.

7.2. Real-Time Aspects

Intelligent monitoring systems must work in real time. In our application, time constraints are weak because ventilation processes are slow (respiratory rate is less than 0,5 Hz; in alarming cases, a response time of 1 or 2 seconds is tolerated). Thus, the sophisticated management of memory (automatic garbage collection) implemented in the current version of Smalltalk (ParcPlace Smalltalk) is sufficient for our application because no unexpected garbage collecting disturbs the working process.

7.3. Uniformity of the representation framework

It is now commonplace in AI applications to mix different orthogonal representation or programming languages. In the Protégé-II project, Shahar et al. (49) used CLIPS to represent planning and temporal abstractions by production rules, as well as the CLIPS object-oriented language extension (COOL) to represent the planning-entity and parameters-properties ontology, and the C language to develop an interface with an external database. Quaglini et al. (44) combined KEE production rules and Common Lisp to represent domain knowledge, as well as an extra language (TellAndAsk) to represent control knowledge.

Linking different representation formalisms introduces a well known "language mismatch" that seriously hampers the declarative aspects of the resulting system, as well as its capacity to be extended. Our representation framework, based on Smalltalk, reduces this language mismatch problem via thorough integration of the underlying language and the rules. Our choice of the Smalltalk language also allows us to benefit from a fast-growing set of standard interfaces to the external world. In particular, the VisualWorks environment (based on the Smalltalk language) provides interface for communication with standard databases, such as Oracle, Sybase, or ObjectStore, from a Smalltalk application, using standard query languages such as SQL.

The programming environment we have described was self-sufficient for the design of our application, and we point out that domain and control knowledge are expressed with the same formalism.

7.4. Factorization and account for limited reuse and context-dependency

Via factorization, inheritance can substantially reduce the quantity of information needed. Bagenholm et al. (1) have developed tools and methods for creating modular independent knowledge units according to the Arden syntax[1]. They have reported that identical medical knowledge is frequently used in many modules. Without inheritance, authors are confronted with redundancy problems.

The importance of "limited reuse" in medical systems encompasses two aspects. First, design and evaluation of medical knowledge-based systems is a long iterative process that implies continuous interaction between computer scientists and health-care personnel. Changes in data processing (new physiological information required and new algorithms developed), the discovery of deficiencies, the development of new strategies for specific medical situations, and user interface adaptation, give rise to a large number of limited software modifications. Second, we doubt that it would be feasible (or worthwhile) to achieve strong standardization of medical expertise because of 1) the impossibility of justifying all details of an expertise based on scientific considerations; 2) the marked domain flexibility of variations in vital parameters in response to changes (adaptation); and 3) the human tendency to reject excessively narrow frames.

Rule-based object-oriented programming offers a suitable representation framework for implementing medical applications because of its adaptability to small changing requirements. It allows the integration and the coexistence of standardized stable concepts and limited clinical team preferences.

[1] The Arden syntax is an attempt to define a standard formalism for representing modular medical knowledge bases.

Our representation language is less powerful than more sophisticated design formalisms such as KADS, KREST, 1st order logic, or conceptual graphs. It has the disadvantage of precluding "design reuse" (5), since design and implementation are combined in a single formalism. However, since only practically achievable functions are designed (and therefore implemented), it does achieve a satisfactory level of actual limited reuse, at the code level.

7.5. Epistemological integrity

Most medical expert systems use frame-based knowledge representation environments, such as KEE or ART. Frames are an object-based formalism that was especially defined to represent knowledge declaratively. In this respect, they are more complex structures than those used in object-oriented programming languages. However, by definition, attached to each frame are several kinds of information, such as information about how to use the frame (35). Thus, the reified world (frames) and the discourse about it (inferences) are closely connected. We believe that, with this approach, the computational level violates the initial epistemological requirements about the clear separation between ontology (the representation of the medical domain) and the different types of inferences that would reflect the proposed epistemological model. We will illustrate this point with an example. Ramoni et al. (45) have argued that a unique inference model working on different ontologies, which represent the conceptual model of entities and relationships composing the domain knowledge, can be used to represent medical tasks. In (44), Quaglini et al. applied this model to hematology for the therapy of anemic patients. Each therapeutic problem and each therapy is represented by a frame. One of the frame's slots contains the name of the rule class to be invoked for assessing the frame's occurrence and testing the frame's appropriateness, respectively. A slot can also contain the name of a metarule class to be triggered. Rules present in slots can use values contained in other slots. Thus, implementation introduces complex links between the levels, instead of maintaining a clear separation. We believe that rules should be separated from therapies 1) to make a clear distinction between a therapy and its use, which may vary; and 2) to facilitate refining or reusing taxonomies of medical entities specific to hematology and general knowledge for therapy planning.

Thus, rule based object-oriented programming offers an interesting alternative to frames in which the separation between levels is enforced. Moreover, simulation - the key principle of object-oriented programming - can be used to evaluate a hypothesis for predicting possible consequences of each plausible therapy plan for the patient (31).

7.6. Separation of control knowledge

The control tasks are essential in patient care activities, especially in real time domains like intensive care or anaesthesia (13), (51). It is the clinical context that allows clinicians to evaluate therapeutic choices, the acceptability of some discrepancies between observed physiological values and normal values, or the total modification of actual therapy. Reification of control objects and inheritance of control knowledge allows to adapt individual diagnoses or to switch carefully from one action plan to another. We have proposed an architecture in which control knowledge is separated from domain knowledge. This separation is particularly useful in medicine where the same (or almost the same) control knowledge can be applied to different domain knowledge modules (e.g., application concerning adverse drug ordering (1)).

Moreover, an important characteristic of our control architecture is that, by construction, meta level for control always prefers control rules (metarules) to domain rules. The basic control loop simply alternates sets of control actions with single domain actions. Hayes-Roth (24) raised several objections to systematic preference of control over domain. At the opposite in blackboard architecture for control, knowledge sources of control and knowledge sources of domain continuously compete, and no specific priority is given *a priori* to control. In practice however, problems caused by this flexibility seem to outweigh beneficial aspects.

8. Conclusion

Because we seek to apply AI techniques to the clinical field, where no consensus in patient care is presently available, the strategies we use must be compared, exchanged, and discussed with the clinical staff of other hospitals with different cultures. Thus, we need an environment where we can integrate, reuse, modify, or refine different strategies and test the performances of each. We propose a framework that combines object-oriented programming and production rules to achieve this goal.

Few systems provide precise descriptions of the implementation level and of practical means used to achieve some forms of reuse. In this paper, we have shown with examples extracted from real clinical expertise 1) how we facilitate reuse of critical software components and introduce small variations, which are useful in medical contexts; 2) how several expertise areas cohabit for a given model; and 3) how the integrity of the conceptual level is preserved.

Object-oriented programming can be used successfully to describe rigorously some parts of clinical activities, allowing panels of clinical experts to achieve a consensus (51). This reinforces the potential interest of the object-oriented paradigm and of production rules association. The development of large, open health care systems might benefit from this approach.

The introduction of asynchronous communications between independent autonomous reasoning modules may be of interest for a framework for real-time intelligent monitoring. Object-oriented programming can be extended to integrate actor language characteristics (for instance (6,28,54) propose actor languages based on Smalltalk-80). We are studying such an extension for the NéoGanesh system.

References

1. Bagenholm, P., Anderskär, K., Gill, H., Jönsson, K. A., et al.: Adding decision support to a clinical information system. Technology and Health Care 1 (1994) 245-251.
2. Barachini, F. and Granec, R.: Productions systems for process control: advances and experiences. AAI 7 (1993) 301-316.
3. Booch, G.: Object-Oriented Design with applications. The Benjamin/Cummings Publishing Company Redwood City (CA) (1991).
4. Brachman, R. J. and Schmolze, J. G.: An overview of the KL-ONE knowledge representation system. Cognitive Science 9 (1985) 171-216.
5. Brender, J. and McNair, P. Watch the system: An opinion on user validation of computer based decision support systems in clinical medicine. *In* "Proceedings of MedInfo'89", (B. Barber, C. Dexian, Q. Dulie and G. Wagner, Eds.), North-Holland, Amsterdam, (1989) pp. 275-279.
6. Briot, J. P. Actalk: a testbed for classifying and designing actor languages in the Smalltalk-80 environment. *In* "Proceedings of ECOOP'89", Eds.), Cook, (1989) pp. 109-130.
7. Chandrasekaran, B.: Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. IEEE Expert 1 (1986) 23-30.
8. Clancey, W. J.: The epistemology of a rule-based expert system: a framework for explanation. Artif Intell 20 (1983) 215-251.
9. Clancey, W. J. and Shortliffe, E. H. Readings in medical artificial intelligence: the first decade (Addison-Wesley, Readings (MA), 1984).
10. Clancey, W. J.: Heuristic classification. Artif Intell 27 (1985) 289-350.
11. Cohen, R., DeLisio, J. and Hart, D.: A declarative representation of control knowledge. IEEE Trans Systems, Man, Cybernetics 19 (1989) 546-557.

12. Cohn, A. I., Rosenbaum, S., Factor, M. and Miller, P. L.: DYNASCENE: An approach to computer-based intelligent cardiovascular monitoring using sequential clinical "scenes". Meth Inf Med 29 (1990) 122-131.
13. Coiera, E.: Intelligent monitoring and control of dynamic physiological systems. Artif Intell in Med 5 (1993) 1-8.
14. Dojat, M., Brochard, L., Lemaire, F. and Harf, A.: A knowledge-based system for assisted ventilation of patients in intensive care. Int J Clin Monit Comput 9 (1992) 239-250.
15. Dojat, M. and Pachet, F. NéoGanesh: an extendable knowledge-based system for the control of mechanical ventilation. *In* "Proceedings of 14th annual international conference of IEEE-EMBS", Eds.), Paris (1992) pp. 920-921.
16. Dojat, M., Harf, A., Touchard, D., Laforest, M., et al.: Evaluation of a knowledge-based system providing ventilatory management and decision for extubation. Am J Respir Crit Care Med (submitted) (1995)
17. Dojat, M. and Sayettat, C.: A realistic model for temporal reasoning in real-time patient monitoring. AAI (to appear) (1995)
18. East, T., Morris, A. H., Wallace, C. J., Clemmer, T. P., et al.: Computerized protocols for the management of arterial oxygenation in patients with ARDS on assist control mechanical ventilation. Am J Respir Crit Care Med (1995) (to appear).
19. Ensing, M., Paton, M., Speel, P.-H. and Rada, R.: An object-oriented approach to knowledge representation in a biomedical domain. Artif Intell in Med 6 (1994) 159-183.
20. Fagan, L. M. Representing time dependant relations in a medical settings. PhD Thesis, Stanford University (CA),(1980).
21. Gardner, R. M.: Patient-monitoring systems. In: E. H. Shortliffe and L. E. Perrault (Eds.) Computer applications in health care. Addison Wesley, Readings, MA, (1990)
22. Goldberg, A. and Robson, D.: Smalltalk-80: The language. Addison-Wesley New York (1989).
23. Gruber, T. R.: Acquiring strategic knowledge from experts. Int J Man-Machine Studies 29 (1988) 579-597.
24. Hayes-Roth, B.: Blackboard architecture for control. Artif Intell 26 (1985) 251-321.
25. Hayes-Roth, F., Erman, L. E., Terry, A. and Hayes-Roth, B. Distributed intelligent control and management: concepts, methods, and tools for developing DICAM applications. Technical Report, KSL Stanford University (CA), KSL 92-33, (1992).
26. Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., et al.: Guardian: a prototype intelligent agent for intensive-care monitoring. Artif Intell in Med 4 (1992) 165-185.
27. Heinsohn, J., Kudenko, D., Nebel, B. and Profitlich, H. J.: An empirical analysis of terminological representation systems. Artif Intell 68 (1994) 367-397.
28. Horslen, S., Capey, A., Casey, A. and Helms, P.: PACE project: object-oriented modelling of paediatric practice. Med Inform 17 (1992) 179-186.
29. Lalonde, W., Thomas, D. and Pugh, J. Actors in Smalltalk Multiprocessor: a case of limited parallelism. Technical Report, Carleton University, Ottawa, Canada, SCS-TR-91, (1986).
30. Lalonde, W. R. and Pugh, J. R.: Inside Smalltalk. Prentice Hall Englewoods Cliffs (NJ) (1990).
31. Langlotz, C. P., Fagan, L. M., Tu, S. W., Sikic, B. I., et al.: A therapy planning architecture that combines decision theory and artificial intelligence techniques. Comput Biomed Res 20 (1987) 279-303.
32. Lau, F. and Vincent, D. D.: Formalized decision support for cardiovascular intensive care. Comput Biomed Res 26 (1993) 294-309.
33. Miksch, S., Horn, W., Popow, C. and Paky, F. VIE-VENT: Knowledge-based monitoring and therapy planning of the artificial ventilation of new-born infants. *In*

"Proceedings of Fourth European Conference on Artificial Intelligence in Medicine Europe", (S. Andreassen, Eds.), IOS, Amsterdam (1993) pp. 218-229.
34. Miller, P. L. Selected topics in medical articial intelligence (Springer-Verlag, New York, 1988).
35. Minsky, M.: A framework for representing knowledge. In: P. Winston (Eds.) The psychology of computer vision. McGraw Hill, New York City, (1975)
36. Morris, A. H.: Paradigms in management. In: M. R. Pinsky and J. A. Dhainaut (Eds.) Pathophysiologic foundations of critical care. Williams & Wilkins, Baltimore (ML), (1993)
37. Musen, M. A.: Dimensions of knowledge sharing and reuse. Comput Biomed Res 25 (1992) 435-467.
38. Nebel, B.: Reasoning and Revision in Hybrid Representation Systems. Springer-Verlag Berlin (1990).
39. Newell, A.: The knowledge level. Artif Intell 18 (1982) 87-102.
40. Pachet, F. and Perrot, J.-F. Rule Firing with Metarules. *In* "Proceedings of SEKE'94", Eds.), Knowledge Systems Institute, Jurmala, (Latvia) (1994) pp. 322-329.
41. Pachet, F. Report on the OOPSLA'94 Workshop on EOOPS. *In* "Proceedings of Addendum to the OOPSLA'94 proceedings", Eds.), ACM SIGPLAN notice, Portland, (OR) (1994) pp.
42. Pachet, F.: On the embeddability of production rules in object-oriented languages. JOOP 8 (1995) 19-24.
43. Patel-Schneider, P. F.: Practical, object-based knowledge representation for knowledge-based systems. Information Systems 15 (1990) 9-19.
44. Quaglini, S., Bellazzi, R., Berzuini, C., Stefanelli, M., et al.: Hybrid knowledge-based systems for therapy planning. Artif Intell in Med 4 (1992) 207-226.
45. Ramoni, M., Stefanelli, M., Magnani, L. and Barosi, G.: An epistemological framework for medical knowledge-based systems. IEEE Trans Systems, Man, Cybernetics 22 (1992) 1361-1375.
46. Rasmussen, J.: Diagnosis reasoning in action. IEEE Trans Systems, Man, Cybernetics 23 (1993) 981-992.
47. Rucker, D. W., Maron, D. J. and Shortliffe, E. H.: Temporal representation of clinical algorithms using expert system and database tools. Comput Biomed Res 23 (1990) 222-239.
48. Rutledge, G. W., Thomsen, G. E., Farr, B. R., Tovar, M. A., et al.: The design and implementation of a ventilator-management advisor. Artif Intell in Med 5 (1993) 67-82.
49. Shahar, Y. Knowledge based temporal abstractions for medical reasoning. PhD Thesis, Stanford University (CA),(1994).
50. Shortliffe, E. H.: The adolescence of AI in Medicine: will the field come of age in the '90s? Artif Intell in Med 5 (1993) 93-106.
51. Uckun, S. Intelligent systems in patient monitoring and therapy management. Technical Report, KSL, Stanford University (CA), KSL 93-32, (1993).
52. Wielinga, B., Schreiber, G. and Breuker, J.: KADS: A modelling approach to knowledge engineering. Knowledge Acquisition 4 (1992) 5-53.
53. Yelland, P. M. Experimental classification facilities for Smalltalk. *In* "Proceedings of OOPSLA '92", Eds.), Vancouver, Canada (1992) pp. 235-246.
54. Yokote, Y. and Tokoro, M. Experience and evolution of ConcurrentSmalltalk. *In* "Proceedings of OOPSLA'87", (N. Meyrowitz, Eds.), Special issue of SIGPLAN notices, ACM, Orlando (USA) (1987) pp. 406-415.