FSS 4082
pp: 1-18 (col.fig.: nil)
PROD. TYPE: COM
ED: DK
PAGN: Usha.N – SCAN: Vinod

Available at WWW.MATHEMATICSWEB.ORG POWERED BY SCIENCE DIRECT
ELSEVIER
Fuzzy Sets and Systems ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮
FUZZY sets and systems
www.elsevier.com/locate/fss

Fuzzy systems based on component software

Ramo Šendelj[a], Vladan Devedžić[b,*]

[a] *Yugoslav Navy, VP 5437 Kumbor, 85340 Herceg Novi, Montenegro, Yugoslavia*
[b] *University of Belgrade, FON - School of Business Administration, Jove Ilića 154, 11000 Belgrade, Yugoslavia*

Received 12 June 2002; received in revised form 25 November 2002; accepted 12 December 2002

Abstract

This paper describes hierarchical modeling of fuzzy logic concepts that has been used within the recently developed model of intelligent systems, called OBOA. The model is based on a multilevel, hierarchical, general object-oriented approach. Current methods and software design and development tools for intelligent systems are usually difficult to extend, and it is not easy to reuse their components in developing intelligent systems. The OBOA model tries to reduce these deficiencies. The model starts with a well-founded software engineering principle, making clear distinction between generic, low-level intelligent software components, and domain-dependent, high-level components of an intelligent system. This paper concentrates on modeling and implementation of fuzzy logic concepts within the hierarchical levels of the OBOA model. The fuzzy components described are extensible and adjustable. As an illustration of how these components are used in practice, a practical design example from the domain of medical diagnosis is shown. The paper also suggests some steps towards future design of fuzzy components and tools for intelligent systems.

Keywords: Fuzzy logic; Fuzzy expert systems; Hierarchical modeling; Software components; Medical diagnosis

1. Introduction

In the general domain of object-oriented software engineering, hierarchical modeling refers to *layered software architectures* [1], in which:

- each component in a system belongs to a certain conceptual *layer* (layers are sets of classes at the same level of abstraction),
- more complex components are designed starting from simpler components, from the same layer or from the lower layers,

* Corresponding author. Tel.: +381-113950856; fax: +381-11461221.
E-mail addresses: ramo@cg.yu (R. Šendelj), devedzic@galeb.etf.bg.ac.yu (V. Devedžić).

0165-0114/03/$ - see front matter
doi:10.1016/S0165-0114(02)00592-4

- A hierarchically organized tree of components that spans across multiple layers can be drawn to represent the architecture of the system.

One particularly important extension of the concept of layered software architecture is the *orthogonal architecture* [18]. In the orthogonal architecture, classes (objects) are organized into layers and threads. Threads consist of classes implementing the same functionality, related to each other by the *dependency* relationship [2,9]. Threads are "vertical", in the sense that their classes belong to different layers. Layers are "horizontal", and there is no *dependency* relationship among the classes in the same layer. Hence, modifications within a thread do not affect other threads. Layers and threads together form a grid. By the position of a class in the architecture, it is easy to understand what level of abstraction and what functionality it implements. The architecture itself is highly reusable, since it is shared by all programs in a certain domain that have the same layers, but may have different threads.

These general concepts have been recently applied to modeling intelligent software systems in the object-oriented way. As a result, a hierarchical model of intelligent systems, called *OBOA* (*OBject Oriented Abstraction*) has been developed [8]. The model encompasses a wide range of knowledge representation methods and inference techniques commonly used today in designing intelligent systems. The purpose of this paper is to describe how the main concepts of fuzzy logic and fuzzy systems, being important modeling techniques and tools in intelligent systems, are supported in the OBOA model.

The purpose of this paper is threefold:

- it shows how the concepts of fuzzy logic and fuzzy systems fit into a more general, object-oriented, hierarchical model of intelligent systems (the OBOA model),
- it explains how design of fuzzy intelligent systems can be facilitated by imposing some hierarchical structure onto the concepts and tools used in the design process,
- it presents an example of how development of practical fuzzy systems can be alleviated using this approach.

The paper is organized as follows. In Section 2, the essence of the OBOA model is described. Section 3 is the central section of the paper. It shows how fuzzy concepts fit into the OBOA model, and presents some design examples. Sections 4 and 5 show examples of current implementation of software components for designing fuzzy systems based on the OBOA model. In Section 6, some informal performance analysis is presented. Section 7 discusses some related research. Finally, Section 8 shows the benefits of this kind of modeling fuzzy systems and directions for future research.

2. Previous work

The OBOA model has been developed over the years and has been described in a number of papers. Its most complete description is presented in [8]. Its development started with the idea of developing a model to support design of intelligent manufacturing systems. However, soon after starting its development we noticed that OBOA can be generalized to modeling intelligent systems regardless of the application area. The general model has then been instantiated to GET-BITS—the

Level of abstraction	Objective	Semantics
Level 1	Integration	Multiple agents or systems
Level 2	System	Single agent or system
Level 3	Blocks	System building blocks
Level 4	Units	Units of blocks
Level 5	Primitives	Parts of units

(a)

Level of abstraction	Dimension			
	D1	D2	...	Dn
Level 1				
Level 2				
Level 3				
Level 4				
Level 5				

(b)

Fig. 1. The OBOA model: (a) the levels of abstraction, (b) dimensions.

model of intelligent tutoring systems [7]. Simultaneously, another line of developing OBOA further has been started—that of ontological engineering of intelligent systems based on OBOA [5,6].

This section illustrates how hierarchical modeling has been included into the OBOA model in order to facilitate design and development of intelligent systems. It also briefly shows how the model supports some well known concepts from the domain of intelligent systems. Other authors' work that has influenced the development of OBOA is briefly surveyed in the section on related research in the end of the paper.

2.1. Levels of abstraction and dimensions in the OBOA model

The OBOA model defines five *levels of abstraction* for designing intelligent systems, Fig. 1a. If necessary, it is also possible to define fine-grained sublevels at each level of abstraction. Each level has associated concepts, operations, knowledge representation techniques, inference methods, knowledge acquisition tools and techniques, and development tools. They are all considered as *dimensions* along which the levels can be analyzed, Fig. 1b. The concepts of the levels of abstraction and dimensions have been derived starting from the orthogonal architecture.

Semantics of the levels of abstractions is easy to understand. In designing intelligent systems, there are *primitives*, which are used to compose *units*, which in turn are parts of *blocks*. Blocks themselves are used to build self-contained *agents* or *systems*, which can be further integrated into more complex systems. For getting a feeling for how OBOA's levels of abstraction correspond to some well known concepts from the domain of intelligent systems, consider the following examples.

Primitives like plain text, logical expressions, attributes and numerical values are used to compose units like rules, frames, and different utility functions. These are then used as parts of certain building blocks that exist in every intelligent system, e.g. classifiers, controllers, and planners. At the system level, we have self-contained *systems* or *agents* like learning systems, scheduling agents, and knowledge-based diagnostic systems, all composed using different building blocks. Finally, at the integration level there are multiagent systems, distributed intelligent systems, and Web-based intelligent systems.

It should be also noted that the borders between any two adjacent levels are not strict; they are rather approximate and "fuzzy". Several concepts related to intelligent systems can be also treated at different levels of abstraction.

Table 1
Some examples of modeling neural networks and genetic algorithms in the OBOA model

Level	Objective	Knowledge representation	Operations	Inference methods	Knowledge acquisition
1	Integration	Hybrid intelligent system			
2	System				Monitoring and acquisition of data,
3	Blocks	Neural networks (NN) Genetic algorithms (GA)		RubNM Evolution GA	NN training GA Reproduction
4	Units	Slab, Layer, Dataset (Training, Test) Population	Get, Put, Propagate, Evaluate, Fitness, MakeChromosome	Propagate, Evaluate Selection	Training, Creating training set and test set
5	Primitives	Neutron, Link PatternOfData, Gene, Chromosome	Get, Put, Activation function Initialization, Mutation, Crossover, Fitness ...		

2.2. *Well-known paradigms and the OBOA model*

As examples of how some well known paradigms and techniques are encompassed by the OBOA model, Table 1 shows how neural networks and genetic algorithms fit in the levels of abstraction from Fig. 1. Note that several entries in the table are left empty. The reason is that Table 1 shows only well known and widely applicable concepts from these two types of intelligent systems.

3. Fuzzy logic and fuzzy systems in the OBOA model

Table 2 shows how fuzzy logic and fuzzy systems fit in the levels of abstraction from Fig. 1. Only some of the concepts from fuzzy logic and fuzzy systems were included in the earlier versions of OBOA [8]. This paper is the first more comprehensive account on how the concepts from fuzzy logic and fuzzy systems are represented in OBOA.

The concepts, operations, methods, etc. at each level of abstraction can be directly mapped onto sets of corresponding components and tools used in designing intelligent systems.

The complexity and the number of these components and tools grow from the lower levels to the higher ones. Consequently, it is quite reasonable to expect further horizontal and vertical subdivisions at higher levels of abstraction in practical applications of the OBOA model for design and development of intelligent systems. Appropriate identification of such subdivisions for some particular types of intelligent systems, such as intelligent tutoring systems and intelligent manufacturing systems, is the topic of our current research.

Table 2
Some examples of modeling fuzzy logic concepts in the OBOA model (after Šendelj, Radović and Devedžić [23] and Šendelj [22])

Level	Objective	Knowledge representation	Operations	Inference methods	Knowledge acquisition
1	Integration				
2	System	Fuzzy logic expert system			Interviews, case studies, learning reasoning strategies
3	Blocks	FuzzyRule, ListFuzzyRule, ListFuzzyVariable, Max-MinInference, Max-Product Inference	addRule, editRule, deleteRule, findRule, addFV, editFV, deleteFV, findFV,	Forward, backward inference	Fuzzy rule training
4	Units	IfPremise, ThenPremise, FuzzyProposition,	GetUnit, AddUnit, EditUnit, Initialization, Create	Max-Min inference, Max-product inference	
5	Primitives	FuzzyVariable, FuzzySet, FuzzyFunction, Hedge, Concentration, Dilation, Indeed, Power, Operation, Union, Complement, Intersection, Defuzzyfication, Relation	Get, Set, Add, Delete,	Defuzzyfication methods,	Manual input, measurement

From the software design point of view, components and tools in Table 2 can be considered as classes of objects. It is easy to derive more specific classes from them in order to tune them to a particular application. The classes are designed in such a way that their semantics is defined horizontally by the corresponding level of abstraction and its sublevels (if any), and vertically by the appropriate key abstractions specified mostly along the concepts. Fig. 2 shows the *FuzzyElement* class hierarchy, represented using the UML notation [9]. Fig. 3 shows more details about a part of the hierarchy from Fig. 2. The “+1” and “+∗” symbols in the figure are attributes of relations between classes and their corresponding objects. For example, a certain fuzzy set corresponds to only one fuzzy variable, whereas multiple fuzzy sets can be defined for a single fuzzy variable. These facts are denoted by putting “+1” next to the FuzzyVariable class and “+∗” next to the FuzzySet class in Fig. 3.

Class interfaces (method procedures) are defined mostly from the operations and inference methods dimensions at each level. The knowledge acquisition and development tools dimensions are used to specify additional classes and methods at each level used for important development tasks of

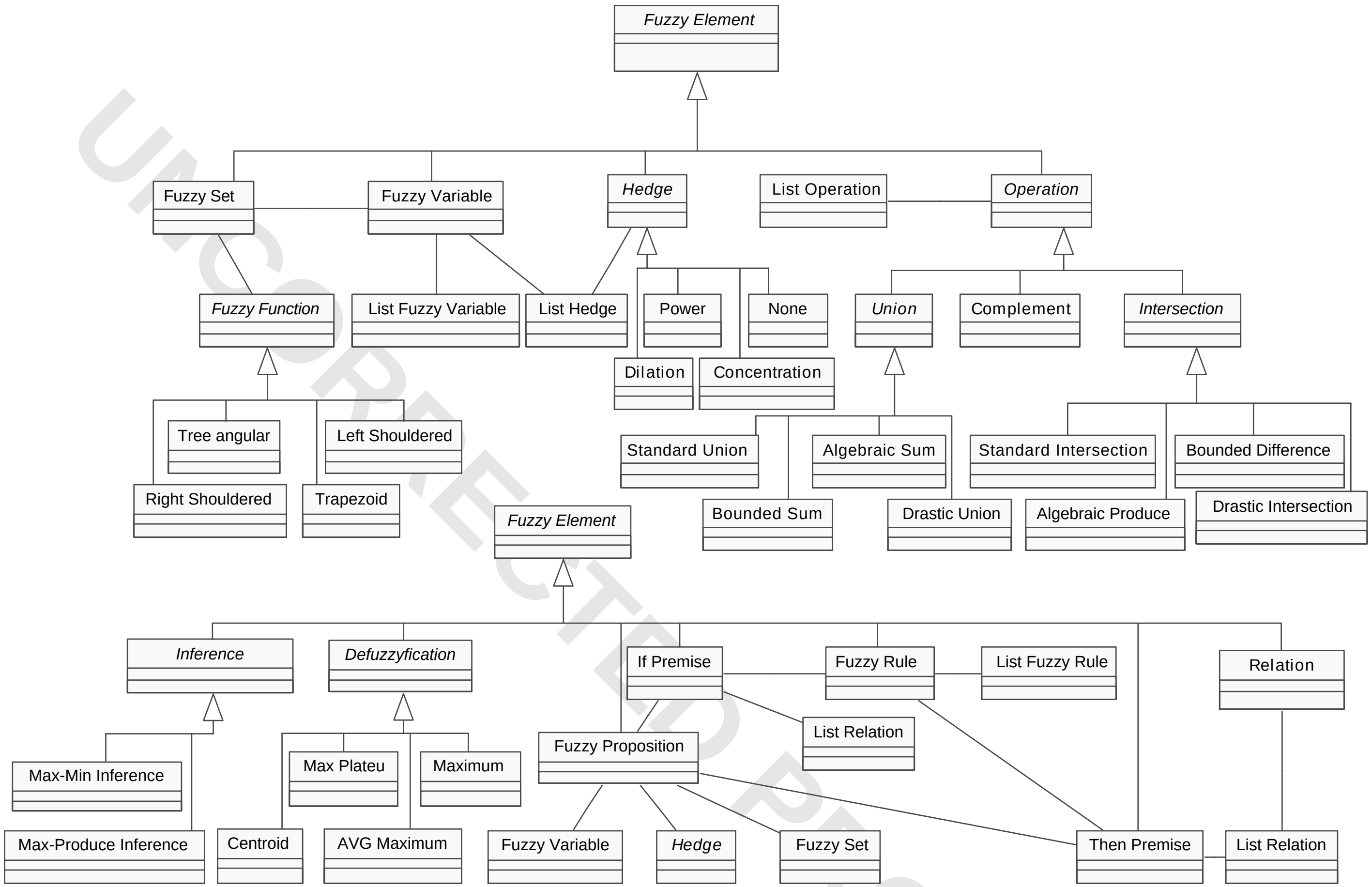

Fig. 2. *FuzzyElement* class hierarchy, represented using the UML notation [9].

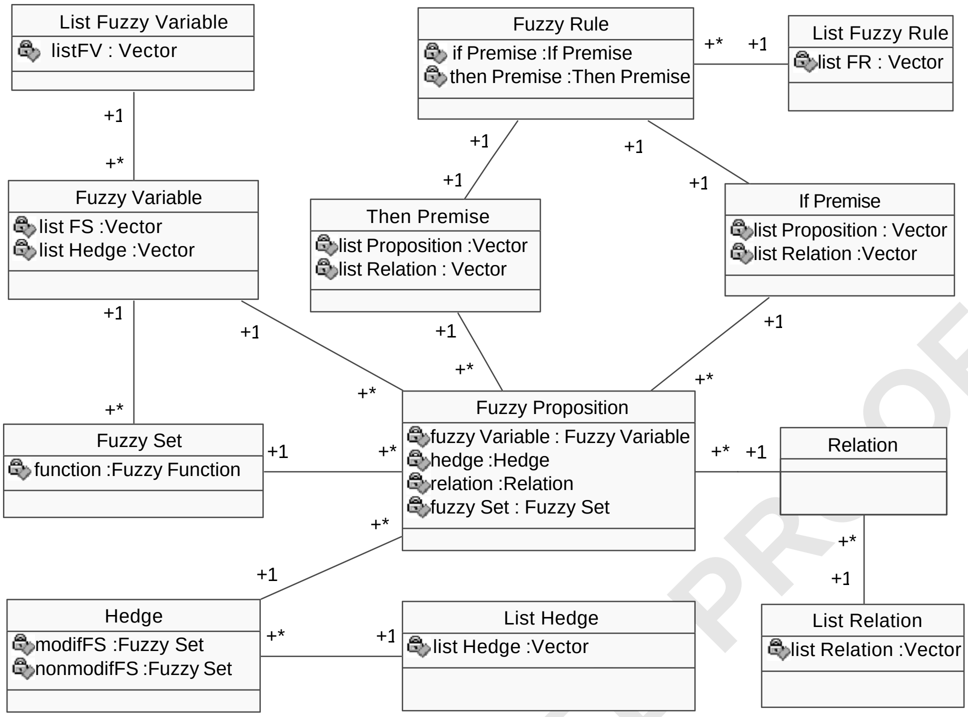

Fig. 3. Detailed view of a part of the class hierarchy from Fig. 2.

knowledge elicitation, learning, and knowledge management. At each level of abstraction, any class is defined using only the classes from that level and the lower ones.

4. Implementation and performance

Using the OBOA model, we have developed interoperable software components [26,27], for the following fuzzy elements: fuzzy variables, fuzzy propositions, fuzzy rules, Max-Min inference and Max-Produce inference. We have developed several other basic fuzzy elements as ordinary classes. Based on the OBOA model and the software components mentioned, we have also designed and implemented a software tool for building fuzzy expert systems, called Fuzzy Expert Systems (FES). We used Java [29] as the implementation language for all the components and classes, as well as for the FES tool.

Press [17] suggests how to test performance of knowledge-based tools like FES, and we used his way. We have created a series of test knowledge bases with different numbers of fuzzy rules in order to analyze FES's characteristics such as the time needed to read the knowledge base from disk, the project's size, and the problem-solving time. Any such a test knowledge base specified an output fuzzy variable (the goal), initial values of input fuzzy variables, and a list of N fuzzy rules, some of which were goal rules (those that set the value of the output variable). Also, test knowledge

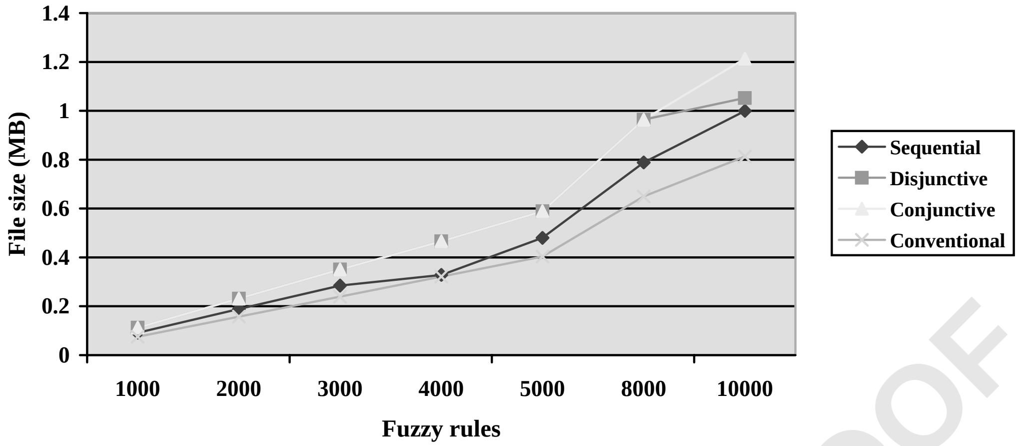

Fig. 4. Knowledge base size for different kinds of fuzzy rules.

bases were created to support three kinds of inference with fuzzy rules:

- *sequential* fuzzy knowledge bases contained fuzzy rules with one fuzzy proposition,
- *disjunctive* fuzzy knowledge bases contained fuzzy rules with two or more fuzzy propositions linked with the OR operator,
- *conjunctive* fuzzy knowledge bases contained fuzzy rules with two or more fuzzy propositions linked with the AND operator.

We wrote a special-purpose program that automatically generates FES fuzzy rules in test knowledge bases and so far have compared characteristics of FES knowledge bases against conventional fuzzy knowledge bases created in Prolog. The tests were run on a PC Windows 2000 system, with 800 MHz Pentium II processor and 128 MB RAM.

Project size increases linearly with the number of fuzzy rules (Fig. 4), but the knowledge-base reading time depends much on the structure of knowledge in the fuzzy knowledge base. Object-oriented OBOA model of fuzzy knowledge bases results in tree-like knowledge structures, which is easier to manipulate and read. The time required to read the knowledge base developed using conventional models typically increases exponentially with the increase of the number of rules (Fig. 5). FES/OBOA is notably superior over conventional models in terms of inference time, which increases linearly in FES up to about 5000 inference cycles (Fig. 6). Inference time becomes exponential with further increase of inference cycles because the impact of operating system and hardware performance becomes more evident.

An important OBOA component built into FES is the *inference optimizer* that reduces the number of rules supplied to the inference engine to perform the inference process, thus considerably reducing the inference time. From the list of all rules in the knowledge base, the inference optimizer selects only those that match the initial values of input fuzzy variables, goal rules, and those needed to carry out that particular inference process—typically 10–15% of all rules. The optimizer also converts disjunctive rules into sets of new equivalent rules.

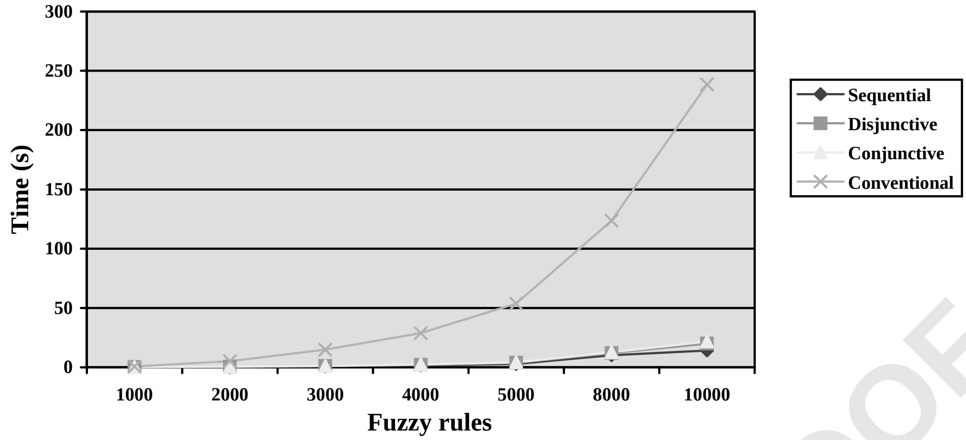

Fig. 5. Reading fuzzy rules from hard disk (reading time).

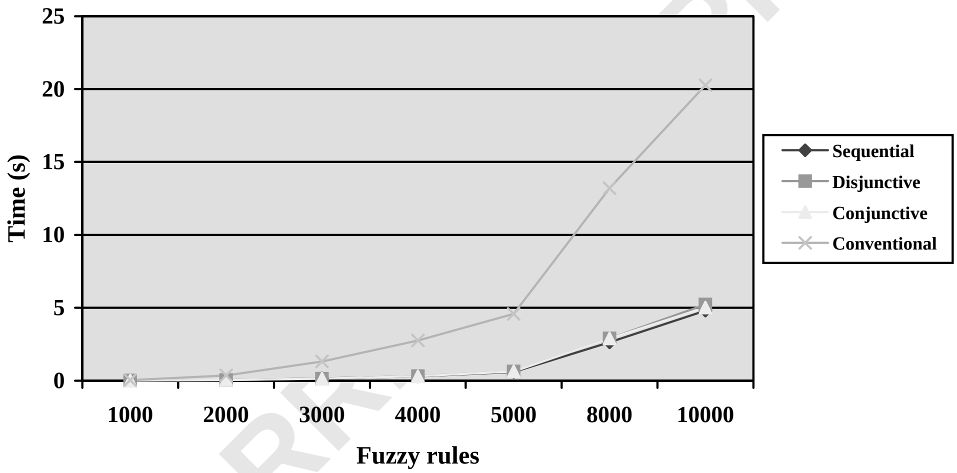

Fig. 6. Inference speed (time).

5. Application example

As a practical example of how our model and the FES tool are used for developing fuzzy expert systems, we show the system we developed for determining the severity of respiratory distress of a patient in an intensive care unit, called Acute Respiratory Distress Syndrome (ARDS).

The lung abnormalities in ARDS are due to diffuse acute lung injury. The lung injury is manifested by several clinical findings, making up the clinical syndrome, which includes: obvious respiratory distress with tachypnea, severe hypoxernia with intrapulmonary shunting (the arterial oxygen pressure decreased, alveolar-arterial oxygen tension difference increased) and diffuse bilateral lung infiltrates on the chest radiograph.

Table 3
Decision-making parameters for ARDS

Phase	Breathing	Rö	PaO_2	$PaCO_2$	A-aDO_2
N	—	—	80–100	35–45	5–10
I	Normal	No changes	70–90	30–40	20–40
II	Mild to moderate tachypnea	Minimal infiltrates	60–80	25–35	30–50
III	Increasing tachypnea	Confluence of infiltrates	50–60	20–35	40–60
IV	Obvious respiratory failure	Generalized infiltrates	35–55	40–55	50–80

ARDS usually develops rapidly and high mortality is still associated with it, in spite of medical technological advance [19]. Hence, detecting ARDS early is of extreme clinical relevance.

Widely used criteria for the early diagnosis of ARDS include:

- clinical aspects of breathing (*Breathing*),
- chest radiograph (**Rö**),
- the arterial partial tension of oxygen (PaO_2, mmHg),
- the arterial partial tension of carbondioxide ($PaCO_2$, mmHg),
- alveolar-arterial oxygen tension difference ($A\text{-}aDO_2$, mmHg).

The progression of changes through phases of ARDS is shown in Table 3. The meanings of symbols in the “Phase” column are:

- N—normal condition of the patient,
- I—the first (the easiest) phase of the respiratory distress (injury and resuscitation),
- II—the second phase of the respiratory distress (subclinical),
- III—the third phase (established respiratory distress), and
- IV—the fourth (the hardest) phase of the distress (severe respiratory failure).

In our design, this medical diagnosis problem is modeled as a fuzzy multicriteria decision-making problem. The patient’s condition is described by a set of symptoms given by numerical values from approximate intervals and by linguistically described features. Fuzzy sets needed for determining the severity of a respiratory distress are modeled using trapezoidal membership functions.

The features *Breathing* and Rö are expressed by linguistic terms. The other features are characterized by approximate numerical intervals of values. They can be interpreted as fuzzy sets of the type “x is approximately in the interval $[b,c]$”, i.e. they can be characterized by ordered quadruple $A=(a,b,c,d)$, a fuzzy trapezoidal number. Such quadruples, i.e. characteristic values of the criteria for determining severity of respiratory distress, shown in Table 3, are represented in Table 4.

Table 4
Fuzzy decision parameters for ARDS

Phase	Breathing	Rö	PaO_2	$PaCO_2$	A-aDO_2
N	—	—	(70, 80, 100, 110)	(30, 35, 45, 50)	(0, 5, 10, 15)
I	Normal	No changes	(50, 70, 90, 110)	(25, 30, 40, 45)	(10, 20, 40, 50)
II	Mild to moderate tachypnea	Minimal infiltrates	(40, 60, 80, 100)	(20, 25, 35, 40)	(20, 30, 50, 60)
III	Increasing tachypnea	Confluence of infiltrates	(40, 50, 60, 70)	(10, 20, 35, 45)	(30, 40, 60, 70)
IV	Obvious respiratory failure	Generalized infiltrates	(30, 35, 55, 60)	(30, 40, 55, 65)	(40, 50, 80, 90)

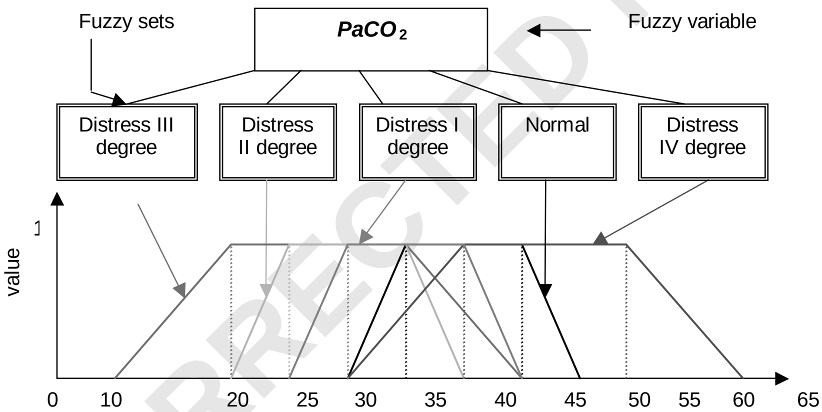

Fig. 7. Fuzzy sets of the fuzzy variable *Arterial partial tension of CO_2*.

For example, the value of the feature $PaCO_2$ for normal condition of the patient is approximately in the interval (80, 100); that can be represented by the fuzzy interval (70, 80, 100, 110). This representation enables the membership degree calculations for each real value of the numerical symptom, in all possible intervals. Membership degrees are taken from the physicians' experience, and as far as the syndrome is considered all the features have the same importance. The maximal degree by which the patient's condition fulfils all the criteria (the features) for the phase is needed, so Bellman-Zadeh's decision-making principle can be applied [30].

Figs. 7–10 show several design and implementation details of our ARDS fuzzy expert system and the FES tool. The underlying software components from OBOA libraries that supported using concepts from fuzzy set theory in building this system are those mentioned in Section 5. Note that

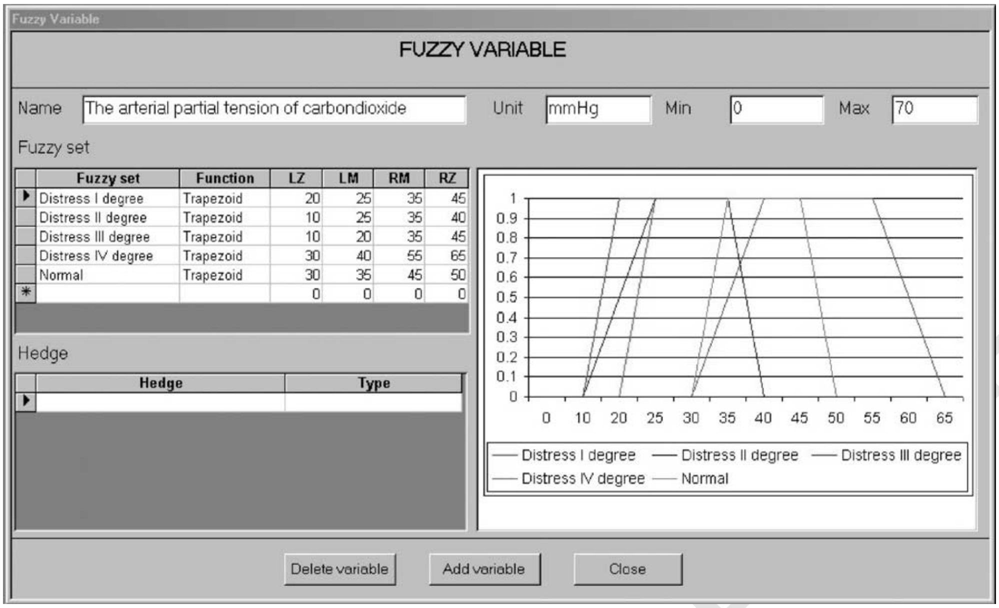

Fig. 8. FES tool—defining new fuzzy variable.

OBOA libraries largely supported plug-and-play approach to design of FES and ARDS fuzzy expert system by providing appropriate GUI components along with those representing concepts from fuzzy sets theory. Fig. 8 shows an example—a dialog box for defining fuzzy variables.

6. Using the OBOA model in practice

Component software is an object-based software movement that subsumes compound document as one example of application interoperability. Component software addresses the general problem of designing system from application elements that were constructed independently by different developers using different languages, tools and computing platforms [26]. The OBOA model also supports design and development of component-based applications. From the component-based software perspective, it should be noted that many of the classes mentioned in Section 5 are actually developed as software components as well.

The OBOA model is supported by a number of design patterns [10] and class libraries developed in order to support building of intelligent systems. In fact, designing and developing an intelligent system based on the OBOA model is a matter of first developing a shell and then using it for development of the system itself. In spite of the fact that this means starting the project *without* a shell, it is a relatively easy design and development process, because of the precisely defined hierarchy among the tools and components, stable and reusable overall layered architecture of OBOA-based systems, as well as the strong software engineering support of the design patterns and class libraries. This is exactly the approach that has been taken in developing our ARDS application. We have started from the OBOA model and its fuzzy elements as specified in Table 2,

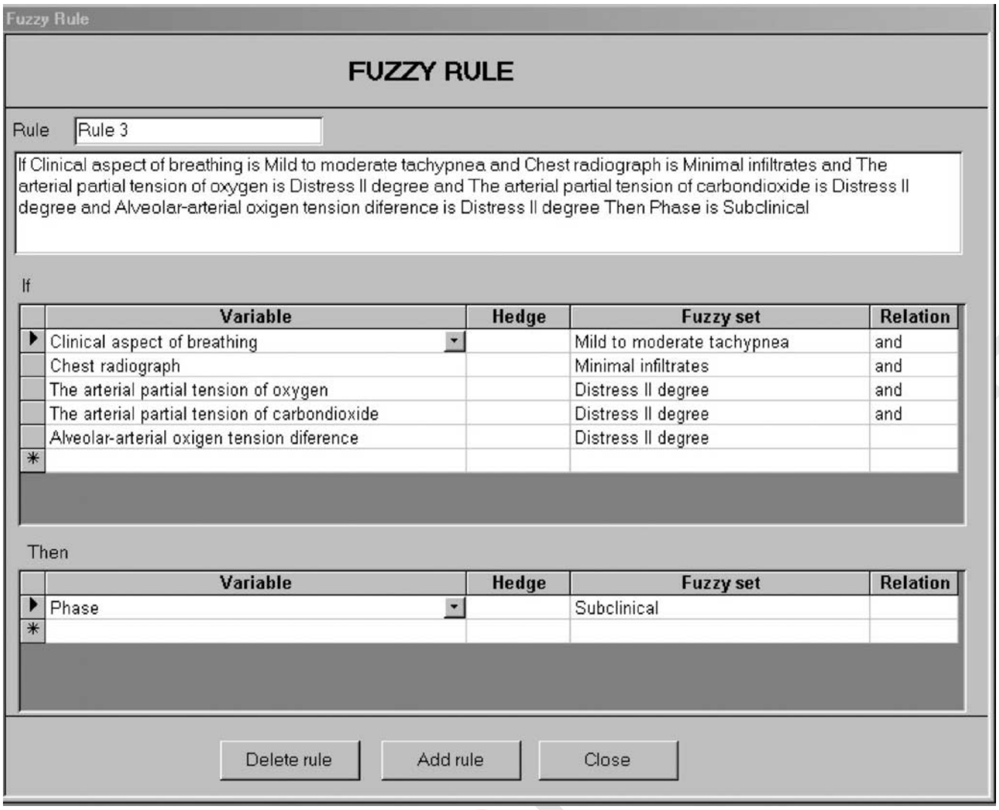

Fig. 9. FES tool-fuzzy rule.

and have designed and developed the corresponding classes and components. Then we used them to assemble the FES tool. Finally, FES has been used as the shell for the development of ARDS system.

How to use, retrieve and integrate OBOA's existing "soft" components in order to build up a fuzzy intelligent application? Software support for building OBOA-based intelligent applications is two-fold. First, a number of OBOA's primitives, units and blocks have been implemented as interoperable software components and placed in a library of reusable interoperable components. These can be used in plug-and-play manner when developing a new system. Section 5 specifies fuzzy elements that have been developed in this way so far. Second, a number of other elements have been implemented as Java classes and application programming interfaces (APIs). These can be (re)used when building new applications, but this requires some programming.

The point is, however, that only the necessary classes and components are retrieved, since OBOA is a model and a framework rather than an integrated development tool. Development of the OBOA-based FES shell for building fuzzy intelligent applications meant putting together only those pieces of software from the relevant class libraries that were really needed for the ARDS application. If any additional class for representing fuzzy concepts was needed, it had to be designed and developed separately and built into the shell. Fortunately, the class hierarchies and design patterns of OBOA

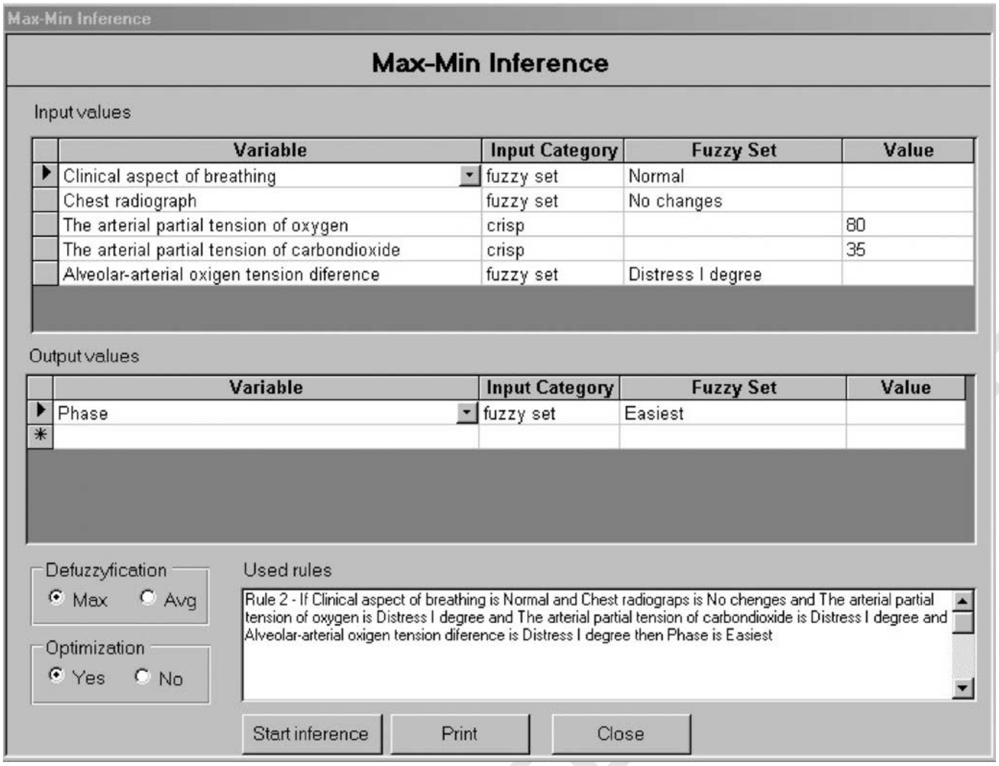

Fig. 10. FES tool—Max-Min inference.

provide a firm ground to start from in such an additional development. Most additional subclasses for representing fuzzy concepts can be derived directly from some of the already existing classes. The classes representing fuzzy concepts in the OBOA model are designed in such a way to specify "concept families" using the least commitment principle: each class specifies only the minimum of attributes and inheritance links. That assures the minimum of constraints for designers of new classes.

As an example, consider the task of adding a new fuzzy element when needed. This task does not require significant changes in the corresponding module of the fuzzy system or the FES tool. It is rather a matter of finding out an appropriate place for the new class along the levels of abstraction and in the class hierarchies representing fuzzy concepts, and specifying a few additional attributes and links. However, it *does* require some expertise in fuzzy logic and systems design. The general strategy we found useful in that sense stems from the one practiced in orthogonal architectures (see Section 1): identify the thread of elements along which to put the new one, and then place the new element in the highest layer that does not require dependency relationships among its classes and the new one.

When developing the FES shell, and then using it for development of an intelligent fuzzy system like ARDS, the shell's options were always only the necessary options. Modifications and extensions are made easily and only in accordance with the application's needs.

Looking at the UML diagrams in Figs. 2 and 3, one may think that OBOA-based development of fuzzy and other intelligent systems does not differ much from any other UML/Java software development. Note, however, that these UML diagrams are shown here only because of the fact that UML notation is well known and widely used. The point is something else entirely—the OBOA model provides an ontology and a framework for building intelligent systems [6], as well as a set of supporting interoperable software components that allow for plug-and-play design in at least some parts of the system. OBOA not only defines a set of APIs for building up "soft" systems such as fuzzy expert systems, neural networks systems, genetic systems—it provides a stable ontological backbone around which such systems can be built.

What does this approach buy us in the context of fuzzy systems modeling/implementation? The answer is, in short—plug-and-play approach whenever possible, as well as a firm engineering foundation and sufficient software support to proceed with modeling/implementation in a cost-effective way when plug and play is not possible immediately. There is a number of ways to model and implement a fuzzy system, in terms of finding a good compromise of contradicting issues such as development effort, cost, flexibility, tool support, and source-code ownership and extension. Another very important development issue is the "ratio" between two fairly distinct kinds of efforts needed to build the system—that of fuzzy modeling of the application domain and that of providing software support to actually implement the model. Ideally, a domain analyst and fuzzy modeler can concentrate on his/her job and rely on complete software support from the tools he/she uses. That approach requires using tools and/or integrated development environments that may be complex, expensive, and difficult to extend when needed. Moreover, new developments in both fuzzy systems and software technology always raise the need for frequent and costly change and upgrades of development tools and environments. At the other end, developers may opt for providing all the necessary software support themselves and thus stay up to date with the latest advances in the field and in technology. However, the software engineering part of building a fuzzy system is then considerably higher. OBOA is somewhere in between the two extremes—it provides a number of pre-built, pre-tested, and easily extensible and upgradeable components and libraries, but not a complex shell or a development environment. It is an ontology-supported stable framework and a collection of easy-to-use components, rather than an integrated software environment that goes through a number of versions over time.

7. Related research

The overall OBOA model was developed with the idea of rooting development of intelligent systems into well understood and stable principles of software engineering and systems engineering, such as hierarchical and orthogonal software architectures, object-orientation, interoperability, component hierarchies, reusability, and component cohesion. OBOA is comprehensive in the sense that it encompasses a number of different intelligent technologies (fuzzy systems, rule-based systems, genetic algorithms, neural networks, intelligent agents) and puts all of them into a common context of intelligent systems. On the other hand, software support for OBOA are libraries of interoperable software components and class libraries. All of them are developed through a thorough ontological analysis of related technologies and concepts.

The authors are not aware of other research efforts that put development of fuzzy systems into a comprehensive context of a similar kind. Still, OBOA was developed with some ideas from other distinct (and different) efforts in mind. One result of such efforts is the Common KADS methodology for development of knowledge-based systems [20]. In OBOA, we deeply appreciate the Common KADS' structured approach to development of knowledge-based systems, as well as its hierarchical modeling of task knowledge, i.e. the knowledge of tasks performed in the organization deploying a knowledge-based system. Another important methodological issue that partially parallels the approaches of OBOA and Common KADS is that of ontological analysis of the application domain. However, OBOA is different from Common KADS in terms of focusing on software architecture and reusability in developing intelligent systems, rather than on organizational models and context or on project management issues, which are extremely important in Common KADS.

OBOA also strives to provide software components and a framework to support development of intelligent systems in much the same way in which other research and development groups do in other fields. For example, within the Object Management Group consortium (OMG) the Manufacturing Domain Task Force (OMG Mfg DTF) has developed a framework and a set of related CORBA objects for building manufacturing systems [14]. In Lockheed Martin, as a part of a DARPA-sponsored project, they have developed a similar set of services and protocols to support concurrent engineering processes throughout the product/process lifecycle by enabling the development of virtual prototypes of products and processes [13].

There is also some other research regarding different aspects of software engineering and fuzzy sets. The most comprehensive collection of published material on software engineering and fuzzy sets from 1990s can be found in [15]. An early work of Lee [12] shows how to use fuzzy sets to evaluate risk in software development throughout the software product life cycle. Zeephongsekul and Xia [31] applied ideas from fuzzy sets to a specific problem of software reliability, that of debugging of software faults. Software engineering of databases using fuzzy sets techniques has also attracted some researchers. An example is a fuzzy generalization of Codd's relational database model, with query evaluation using linguistic modifiers [21].

More recently, some efforts in software engineering using concepts from fuzzy sets can be found in the work of Cross and Firat [4]. They have developed a fuzzy object data model for representing knowledge underlying geographical information systems. Their approach is different from OBOA, but is interesting from the perspective of integrating their fuzzy object data model with commercial, of-the-shelf tools—an expert system shell and a commercial object-oriented database system. Similarly, Slonim and Schneider [24] used fuzzy-valued properties to represent cases in case-based reasoning systems. Pedrycz and Sosnowski [16] have studied the design of decision trees using fuzzy sets and have applied it to quantifying complexity of software systems in the framework of decision trees. There is also a fuzzy logic-based approach to identification of potentially error-prone software components, which an important issue in software inspection [25]. The work of Chen [3] included development of a new algorithm to evaluate the rate of aggregative risk in software development using fuzzy set theory under the fuzzy group decision making environment. Finally, two recent approaches of interest for further development of OBOA are described in [11] and [28]. Huang's work is important in terms of using fuzzy sets within an intelligent agent framework to carry out modular design of products to meet a customer's fuzzy requirements using modules that come from suppliers that are geographically separated and operate on differing computer platforms. He stresses the need to consider intelligent agents as software components of a complex system, which roughly corresponds to the ideas of Levels 1 and 2 in OBOA (see Fig. 1). Wang and Lin strive to develop a multi-criteria group decision making model based on fuzzy set theory to select configuration items for software development. Their work is important for OBOA since a similar approach can be used in selecting existing OBOA components and classes to configure an OBOA-based intelligent system. However, we still did not use that approach in practice.

8. Conclusions

Hierarchical design of fuzzy-logic concepts of intelligent systems, presented in the paper, allows for easy and natural conceptualization and design of a wide range of intelligent applications, due to its object-oriented approach. It suggests only general guidelines for developing fuzzy intelligent systems, and is open for fine-tuning and adaptation to particular applications. Fuzzy intelligent systems developed using this model are easy to maintain and extend, and are much more reusable than other similar systems and tools.

The model is particularly suitable for use by developers of software environments (shells) for building fuzzy systems. Starting from a library of classes and interoperable software components for fuzzy logic concepts and control needed in the majority of fuzzy systems, it is a straightforward task to design additional fuzzy logic classes needed for a particular fuzzy system shell. Moreover, the model also supports development of component-based intelligent systems, which have started to attract increasing attention among the researchers in the field.

Further development of support for fuzzy logic concepts in the OBOA model is concentrated on development of appropriate classes in order to support a number of different fuzzy systems. The idea is that the system developer can have the possibility to select fuzzy tools from a predefined palette, thus adapting the shell to his/her own design preferences. Such a possibility would enable experimentation with different fuzzy tools and their empirical evaluation.

As the technology is making progress in abilities to deliver knowledge to the desktops of practicing clinicians, especially by the World-Wide Web, the universal issues of reconciliation and delivering of relevant medical knowledge to practitioners using the Internet technology are getting more and more important. Then, the Java programming language is the candidate for developing distributed intelligent applications available on a variety of computing platforms, in order to enable users to use the (multimedia) information they have access to. The presented fuzzy model contributes to these general developments by enabling distribution of information that is not so well structured.

References

[1] D. Batory, S. O'Malley, The design and implementation of hierarchical software systems with reusable components, ACM Trans. Software Eng. Methodol. 1 (4) (1992) 355–398.

[2] G. Booch, J. Rumbaugh, I. Jacobson, Unified Modeling Language User's Guide, Addison-Wesley, Reading, MA, 1998.

[3] S.-M. Chen, Fuzzy group decision making for evaluating the rate of aggregative risk in software development, Fuzzy Sets and Systems 118 (1) (2001) 75–88.

[4] V. Cross, A. Firat, Fuzzy objects for geographical information systems, Fuzzy Sets and Systems 113 (1) (2000) 19–36.

[5] V. Devedžić, Ontologies: borrowing from software patterns, ACM Intell. Mag. 10 (4) (1999) 14–24.

[6] V. Devedžić, Understanding ontological engineering, Comm. ACM 45 (4ve) (2002) 136–144.
[7] V. Devedžić, Lj. Jerinić, D. Radović, The GET-BITS model of intelligent tutoring systems, J. Interactive Learning. Res. 11 (3/4) (2000) 411–434 (Special Issue on Intelligent Systems/Tools in Training and Lifelong Learning).
[8] V. Devedžić, D. Radović, A framework for building intelligent manufacturing systems, IEEE Trans. Systems, Man, Cybernet. Part C—Appl. Rev. 29 (3) (1999) 422–439.
[9] M. Fowler, K. Scott, UML Distilled, 2nd Edition, Addison-Wesley, New York, 2000.
[10] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.
[11] C.-C. Huang, Using intelligent agents to manage fuzzy business processes, IEEE Trans. Systems, Man, Cybernet. Part A: Systems Humans 31 (6) (2001) 508–523.
[12] H.-M. Lee, Group decision making using fuzzy sets theory for evaluating the rate of aggregative risk in software development, Fuzzy Sets and Systems 80 (3) (1996) 261–271.
[13] Lockheed Martin Artificial Intelligence Center, SBD Systems Design Paper, 1997, available at: http://sbdhost.parl.com/sbd_paper.html.
[14] Object Management Group, Standardizing Business Components for the Manufacturing Enterprise, Document mfg/97-10-05, Object Management Group, Framingham, MA, 1997.
[15] W. Pedrycz, J.F. Peters (Eds.), Computational Intelligence in Software Engineering (Advances in Fuzzy Systems, Applications and Theory, vol. 16), World Scientific, Singapore, 1998.
[16] W. Pedrycz, Z.A. Sosnowski, The design of decision trees in the framework of granular data and their application to software quality models, Fuzzy Sets and Systems 123 (3) (2001) 271–290.
[17] L. Press, Expert system benchmarks, IEEE Expert 4 (1) (1989) 33–44.
[18] V. Rajlich, J.H. Silva, Evolution and reuse of orthogonal architecture, IEEE Trans. Software Eng. 22 (2) (1996) 153–157.
[19] D. Šaletic, D. Velaševic, An extended model of one category of fuzzy expert systems, Proc. SYMOPIS '99 Conf., Herceg Novi, Yugoslavia, 1999, pp. 318–324.
[20] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans, W. Van de Velde, CommonKADS: a comprehensive methodology for KBS development, IEEE Expert 9 (6) (1994) 28–37.
[21] M. Šeda, J. Dvorak, Measuring fuzzy query responses in similarity based models, in: P. Navrat, H. Ueno (Eds.), Knowledge-Based Software Engineering, IOS Press, Amsterdam, 1998, pp. 266–269.
[22] R. Šendelj, Representational hierarchy of fuzzy logic concepts in the OBOA model, Proc. AIE/EIA 99 Conf. Cairo, Egypt, 1999, pp. 234–243.
[23] R. Šendelj, D. Radović, V. Devedžić, Java class for knowledge representation, Proc. SYMOPIS '98 Conf. Herceg Novi, Yugoslavia, 1998, pp. 337–340.
[24] T.Y. Slonim, M. Schneider, Design issues in fuzzy case-based reasoning, Fuzzy Sets and Systems 117 (2) (2001) 251–267.
[25] S.S. So, S.D. Cha, Y.R. Kwon, Empirical evaluation of a fuzzy logic-based software quality prediction model, Fuzzy Sets and Systems 127 (2) (2002) 199–208.
[26] V. Szyperski, Component Software: Beyond Object-Oriented Programming, Addison-Wesley, Reading, MA, 1998.
[27] S. Vinoski, CORBA: integrating diverse applications within distributed heterogeneous environments, IEEE Comm. Mag. 14 (2) (1997) 28–40.
[28] J. Wang, Y.-I. Lin, A fuzzy multicriteria group decision making approach to select configuration items for software development, Fuzzy Sets and Systems (2003), in press.
[29] R. Winder, G. Roberts, Developing Java Software, 2nd Edition, Wiley, New York, 2000.
[30] L.A. Zadeh, QSA/FL-qualitative systems analysis based on fuzzy logic, Proc. AAAI Symp. on Limited Rationality, Stanford, 1989, pp. 111–114.
[31] P. Zeephongsekul, G. Xia, On fuzzy debugging of software programs, Fuzzy Sets and Systems 83 (2) (1996) 239–247.