A General Architecture for Intelligent Tutoring of Diagnostic Classification Problem Solving

Rebecca S. Crowley, MD, MS[1,2,3] and Olga Medvedeva MS, MS[1]

[1] Center for Pathology Informatics, University of Pittsburgh School of Medicine, Pittsburgh PA
[2] Center for Biomedical Informatics, University of Pittsburgh School of Medicine, Pittsburgh PA
[3] Intelligent Systems Program, University of Pittsburgh, Pittsburgh PA

ABSTRACT

We report on a general architecture for creating knowledge-based medical training systems to teach diagnostic classification problem solving. The approach is informed by our previous work describing the development of expertise in classification problem solving in Pathology. The architecture envelops the traditional Intelligent Tutoring System design within the Unified Problem-solving Method description Language (UPML) architecture, supporting component modularity and reuse. Based on the domain ontology, domain task ontology and case data, the abstract problem-solving methods of the expert model create a dynamic solution graph. Student interaction with the solution graph is filtered through an instructional layer, which is created by a second set of abstract problem-solving methods and pedagogic ontologies, in response to the current state of the student model. We outline the advantages and limitations of this general approach, and describe it's implementation in SlideTutor – a developing Intelligent Tutoring System in Dermatopathology.

INTRODUCTION

An important potential application of knowledge-based systems in Medical Informatics is the training of future medical professionals. Despite significant work in medical knowledge engineering and decision-support, advances in these areas are rarely applied in computer-based medical education. One potential reason is the absence of general architectures or frameworks for incorporating medical knowledge bases in larger instructional systems. Just as domain knowledge is necessary, but not sufficient, for competent teaching, domain knowledge systems are necessary, but not sufficient for competent, knowledge-based teaching systems. We outline a general architecture for development of knowledge-based medical training systems, and describe the application of this architecture in SlideTutor [1], a developing training system in Dermatopathology. The empirical foundation for SlideTutor is our ongoing cognitive analysis of visual diagnostic classification [2]. The approach incorporates aspects of intelligent tutoring system (ITS) design [3], as well as methods of knowledge modeling [4], establishing a general architecture for instruction in classification problem solving.

BACKGROUND

ITS are adaptive, instructional systems that seek to emulate the well known benefits of one-on-one tutoring when compared to other instructional methods. Many ITS have been shown to improve student performance beyond classroom instruction, and close to what can be achieved with a human tutor [5]. Model tracing ITS (MT-ITS) are a subtype of ITS that guide the student through the problem space, correcting errors in the intermediate steps and offering hints specific to the current problem state [6]. Dialogue systems are another kind of ITS that provide an ongoing conversation about problem-solving, in which the dialogue may be student-initiated, system-initiated, or mixed-initiative. Any ITS typically includes four basic components: expert module, student model, pedagogic knowledge and interface [3]. The expert module provides a model of expert performance against which the student is measured. Across multiple problems or cases, ITS dynamically model the student. Assessment of the student state may influence the system's pedagogic decisions about tutorial strategy, case selection and curriculum sequencing.

Only a handful of medical ITS have been developed [7-10]. Of these, the GUIDON project [10] is most directly relevant. GUIDON used 200 tutorial rules as well as MYCIN's 400 domain rules to teach medical students to learn to identify the most likely causative organism in cases of infectious meningitis and bacteremia. Before a case was presented, consultation with MYCIN was used to generate an AND/OR tree representing Goals (OR nodes) and Rules (AND nodes). GUIDON then used the AND/OR tree to structure the discussion with the student. Interaction used a mixed-initiative method of dialogue. One of the central advances of GUIDON was the separation of domain knowledge from pedagogic knowledge. However, by current standards, the scope of domain knowledge was quite small. Consequently, GUIDON's feedback was only relevant to a very small part of the domain.

GENERAL ARCHITECTURE

The central assumption in our architecture is that human cognition is modeled by production rule systems. This assumption is based on a framework of human cognition that has accumulated abundant empirical validation [11]. The commitment to a rule-based expert model is needed to achieve a teaching system that is congruent to human cognition and capable of providing rapid student feedback. However, the rule system architecture can be limiting. Like GUIDON, almost all existing ITS use production rules that are specific to the domain knowledge they teach. In abstract, procedural, and tightly circumscribed domains such as Algebra, Physics, or Programming this poses no specific disadvantages. In contrast, development of medical tutoring systems requires highly complex, large declarative knowledge structures, which may evolve over time. To address this requirement, we utilize a common knowledge engineering method - abstract problem solving methods and ontologies. The architecture we propose (Figure 1) is consistent with the constraints of the UPML Component Model [12], resulting in an ITS architecture that is modular and re-usable, but also cognitively plausible.

The expert model is composed of domain model ontology, domain task ontology, and abstract problem solving methods. The domain model ontology represents the domain knowledge used to solve problems. The domain task ontology represents the goals of the problem-solving process. In combination with the case data, the abstract problem-solving methods create a dynamic solution graph which student actions are tested against. The pedagogic model is composed of a separate set of component ontologies and problem-solving methods including a pedagogic ontology, pedagogic task ontology and abstract problem solving methods. The pedagogic model ontology represents the pedagogic knowledge used to teach problem-solving. The pedagogic task ontology represents the goals of the instructional process. In combination with data reflecting the current state of the student model, the pedagogic model creates a highly flexible and context-specific instructional layer between the student interface and the dynamic solution graph.

Advantages of this architecture include: (1) Much larger expert models are possible. Students have a tendency to wander far afield in any problem space. Large models enable targeted feedback that guides students back to the solution path. (2) Many diagnostic medical problems have deep, structural task similarities – searching, identification, hypothesis formation and testing, etc. A general architecture can be reused with many domain ontologies, pedagogic models, or student interfaces. (3) Knowledge in the ontologies can be easily modified or expanded without altering the underlying code.

IMPLEMENTATION IN SLIDETUTOR

SlideTutor [1] is a developing medical ITS, that uses this general architecture. The expert model of SlideTutor is composed of a domain task ontology instantiated for visual classification, a domain model instantiated in Inflammatory Skin Diseases, and the abstract problem solving methods for generating the dynamic solution graph. The domain task ontology, and abstract problem solving methods for the system have been completed. The domain model is instantiated for one of eleven algorithms in Inflammatory Skin Diseases [13]. On the pedagogic side, we have created and partially instantiated the pedagogic model, and built an initial set of the abstract problem-solving methods for creating the instructional layer. But flexibility in the instructional layer requires a functional student modeling system, which is not yet implemented. Therefore, we are currently using a default set of pedagogic goals.

SlideTutor is implemented as a client-server application in the Java and Jess programming languages. Ontologies are developed with Protégé-2000 (protege.stanford.edu) and problem-solving methods are written and executed in Jess – a Java production rule system (herzberg.ca.sandia.gov/jess/). Our Virtual Microscope system extends a commercial software package for web-based image panning and zooming (www.xippix.com). A prototype student interface for use with the system can be downloaded from the project homepage at (slidetutor.upmc.edu) using Java Web Start. (java.sun.com/products/javawebstart/). The student interface combines a virtual slide viewer and diagrammatic reasoning interface (Figure 2). Users can search an example whole slide image using a virtual microscope, point to features of importance in the image, and construct a graphical argument for one or more diagnoses. Each user step in the interface is evaluated against SlideTutor's expert model, and appropriate feedback is generated. For programming and debugging the dynamic solution graph, we extended JGraph (www.jgraph.org) to provide a visualization of the evolving problem state (Figure 3).

KNOWLEDGE REPRESENTATION

SlideTutor is currently instantiated in the sub-domain of Inflammatory Diseases of Skin, an area of the domain that is highly algorithmic. Instances are specific to one of the eleven diagnostic algorithms [13]– Subepidermal Vesicular Dermatitis. Further work is planned to represent disease

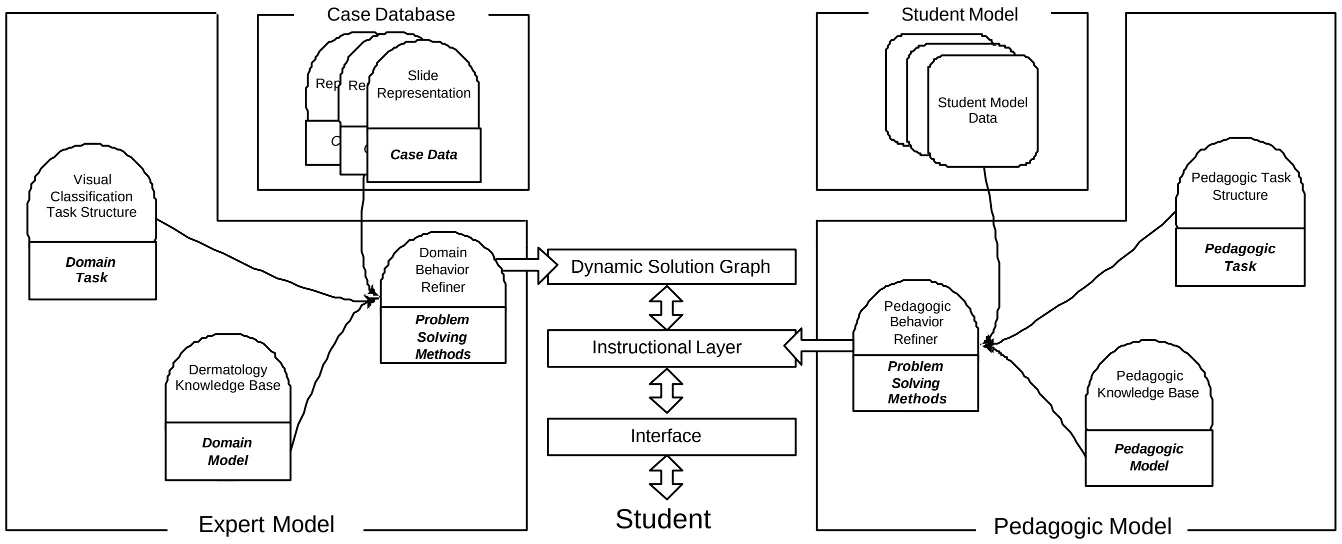


Figure 1. General Architecture for Intelligent Tutoring of Classification Problem Solving

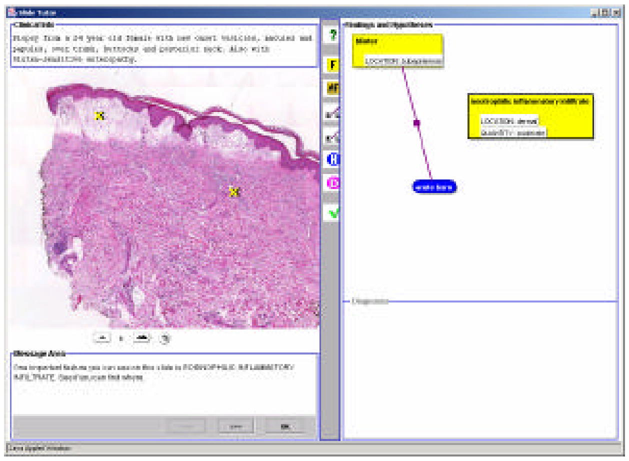


Figure 2. Current node-and-arc reasoning interface.

entities in the remaining ten algorithms. Details of SlideTutor's expert model knowledge representation have been previously described [1]. Briefly, to generate the dynamic solution graph, abstract problem solving methods are applied to three sets of working memory elements (frames): ***domain model*** (dermatopathology knowledge-base), ***case data*** (slide representation for a given case), and ***domain task model*** (visual classification task structure indicating the sequence of possible task actions).

Domain Model. The class concepts and relationships of the domain model ontology are general, and apply widely throughout Pathology and other areas of medicine in which classification is feature-based. Our representation slightly extends the ontology for classification problem solving described by Motta et al [14]. Diseases are hierarchically represented. Any disease may belong to more than one class (multiple inheritance). Diseases have any number of FEATURE_SPECIFICATION instances representing the distinct histologic patterns that are to be learned, which are built up from features, attributes, and values. Instances of FEATURE represent distinct perceptual primitives of visual entities (such as blister) that form the "atoms" of visual feature recognition. Instances of the class ATTRIBUTE_VALUE_LIST are composed of sets of ATTURBUTES and VALUES, and represent the additional cognitive steps for refining these features (such as the distinction of a blister's location relative to the epidermis as sub-epidermal or intra-epidermal).

Case Data. Each instance of SLIDE represents the location and content of each lesion and feature encoded in a single slide. ATTRIBUTE, VALUE and FEATURE are the same classes used in the Knowledge-Base. Instances of FEATURE and unique sets of attributes and values are represented as instances of class OBSERVABLE. Instances of the class OBSERVABLE model values specific to the given case as opposed to the attribute ranges of FEATURE_SPECIFICATIONS. SLIDE representations are created using a Slide Authoring Tool.

Domain Task Model. The domain task model represents goals to be achieved including (1) finding a focal lesion, (2) identifying visual features, (3) modifying features with attributes and values, (4) asserting a hypothesis, (5) linking evidence to hypothesis, (6) setting a goal to find a feature that distinguishes between multiple hypotheses, and (7) making a diagnosis. Instantiation of the domain task model occurs at run-time, dependent on the case and student, and will ultimately be subject to decisions of the pedagogic model. Goals contain priorities that will ultimately be set by the pedagogic model.

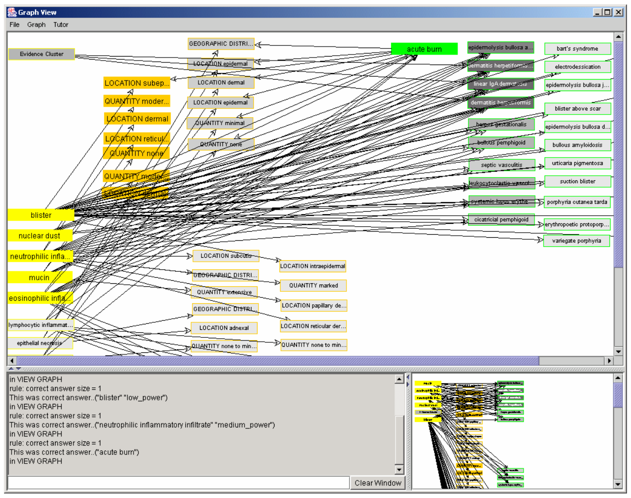
DYNAMIC SOLUTION GRAPH

The dynamic solution graph (DSG) is a directed acyclic graph, designed to generate valid paths through the problem space, based on the case data, domain model, and domain task model. A graph representation supports many useful features for a teaching system. It enables both forward reasoning and backwards reasoning, and can express negated nodes as well as negated relationships. The DSG is capable of generating the entire space, but any individual state of the DSG indicates only the current problem state and all valid next-steps. Any valid next-step events propagate iteratively through the abstract nodes until all the current pedagogic goal requirements are met. The actions that occur in response to a triggered event alter the solution graph, specific to the type of goal node. This function is encapsulated in the unique behavior structures for each different type of node. The behavior of DSG is dependent on the order of input events applied to the current problem state, therefore forward and backwards reasoning generate different graphs.

Our implementation uses cluster nodes to represent the superset of nodes of the same type. Arcs between clusters and other nodes express an integrated relation between the state of the cluster's elements and nodes outside the cluster. For example, a single piece of evidence can be associated with different FEATURE_SPECIFICATIONS of the same DISEASE. Thus, the evidence cluster contains all potential FEATURE_SPECIFICATIONS, as the problem advances towards a solution state. At the conclusion of successful problem-solving, only one FEATURE_SPECIFICATION remains for each valid disease hypothesis. Furthermore, evidence clusters allow two potential pedagogic strategies for feedback about evidence-hypothesis relationships. The system can permit hypothesis formation based on a single piece of evidence, or require that a hypothesis be consistent with all accumulated evidence.

Thus, the decision to use a dynamic graph was based on multiple factors. The DSG (1) limits the size of working memory for a solution space that can be extremely large, (2) allows disambiguation as the problem state changes and (3) provides flexibility in pedagogic feedback as well as the potential to completely change pedagogic strategy based on individual student actions.

For programming and debugging purposes, we extended JGraph to "walk" the dynamic graph (Figure 3). The DSG debugger can be initialized to any problem state. Clicking



```
in VIEW GRAPH
rule: correct answer size = 1
This was correct answer..("blister" "low_power")
in VIEW GRAPH
rule: correct answer size = 1
This was correct answer..("neutrophilic inflammatory infiltrate" "medium_power")
in VIEW GRAPH
rule: correct answer size = 1
This was correct answer..("acute burn")
in VIEW GRAPH
```

Figure 3. Visualization of DSG with DSG debugger.

on nodes propagates events and updates the graph to the next problem state, simulating student actions.

INSTRUCTIONAL LAYER

The purpose of the instructional layer is to provide feedback to students as they solve problems. In SlideTutor, the instructional layer is implemented as a model-tracer [6] – a system that provides step-by-step feedback and hints based on it's expert model. However, the modularity of our design permits the same expert model to be used with any instructional layer (for example a dialogue system or a critiquing system) as well as any interface.

Our model-tracer matches student actions against the DSG and generates an appropriate response. When student actions reproduce reasonable paths through the problem-space, the state of the DSG advances with each *correct action*, but the instructional layer takes no action. When the student takes an *incorrect action*, the instructional layer analyzes the student action and determines which "bug" message to return to the student. When the student is unsure what to do and asks for a *hint*, the instructional layer queries the DSG to determine the valid next-step with the highest priority.

Bug and Hint text messages are instances of template hints from the Pedagogic Knowledge Base, into which values specific to the student action and current problem state are inserted. In addition to presenting context-specific text, bug and hint messages may also direct the student's visual attention - for example by moving the viewer to the area with the lesion, or annotating a particular feature on the whole slide. Hint messages are hierarchical. Early hints offer general advice, but subsequent requests return specific instructions about appropriate next actions.

The current implementation of the instructional layer includes only the functionality required to provide the three basic model-tracing facilities described above. In future work, we will significantly expand the role of the instructional layer, develop (or modify existing) general classes for teaching, create instances appropriate to the teaching of diagnostic classification problem solving, and write additional abstract problem solving methods that configure the instructional layer specific to the current state of the student model. We intend to create an instructional layer that can responsively adapt to changing student needs, by changing the pedagogic approach (e.g. to a case based approach), altering the interface (e.g. to an algorithmic interface), changing hint modalities (e.g. when students respond better to visual or to textual hints) and permitting more advanced student behaviors (e.g. step-skipping, incomplete articulation).

EXAMPLE INTERACTION

To clarify the functions and relationships of these components, we trace the interaction between a hypothetical student and SlideTutor for three steps in a problem – a *correct action*, an *incorrect action* and a *hint* request. A case of Dermatitis Herpetiformis is presented to the student as an unknown. At the beginning of the session, instances from the Dermatopathology Knowledge Base and the Slide Representation of that case are asserted as Jess facts. The problem solving methods of the expert model, create a new set of Jess facts based on the Jess templates (classes) defined in the Visual Classification Task Model, and construct a graph of the present problem state and valid-next-steps. As the problem starts, the only valid-next-step is to find the area of the slide with the lesion on it. Our student successfully traverses this area with the virtual microscope. The instructional layer takes no action. The system registers this *correct action* and problem-solving methods generate the next state of the solution graph. The new valid-next-steps for this novice student include only the identification of evidence in the slide, because the pedagogic model requires complete articulation of evidence.

The student finds an area of discontinuity between the epithelium and the dermis, points to it in the image, and selects "epidermal necrosis" as the feature. This step is an *incorrect action*. The DSG state does not change. The instructional layer determines that the location the student selected in the image is associated with feature "blister" – not "epithelial necrosis" and responds with the message: "There is an important feature in that area but it's not <epithelial necrosis>" by inserting the current user input (in brackets) into the specific bug message for this situation. The template text of this bug message originates from the pedagogic knowledge base. The system also indicates the error by creating a finding icon in the diagrammatic reasoning space, which is marked in red. The student cannot do anything with this icon except to delete it.

The student does not know what to do and asks for a *hint*. The DSG state does not change. The instructional layer responds by moving the viewer to the correct magnification and field and indicating that this is the area where the feature is found. Additional hint requests generate: (1) a text message there is an epidermal change, (2) a text message indicating that this is the feature in question accompanied by annotation of the image, and eventually (3) a text message telling the student that this is a Blister. Once the Blister icon has been created, the student must also specify that the Blister has a location that is sub-epidermal (attribute-value) because the distinction of sub-epidermal from intra-epidermal is critical in this classification algorithm.

The current default state of the pedagogic model requires that any hypothesis must be preceded by at least one piece of evidence. Therefore, the next DSG state will be the first time that possible hypotheses based on that evidence appear as valid-next-steps in the graph. The evidence cluster supports hypotheses that match all of the evidence. So hints at this point can be directed towards the best hypotheses (supported by the cluster), even though the system may accept any hypothesis that is consistent with one piece of found evidence. Once a hypothesis is created (Figure 2), the solution graph contains nodes for all valid support and refute actions that link found-evidence to hypotheses (Figure 3). Additionally, it contains every feature/attribute/value combination valid for any FEATURE_SPECIFICATION of that hypothesis. This models the "backwards reasoning" component. In this way, the system can eventually provide feedback that reinforces efforts to find these features, especially those that distinguish among the current hypotheses.

LIMITATIONS

One of the primary limitations of the architecture we propose is that is not well suited for domains where there are no clear prototypical instances or schemas. All possible combinations of the evidence cannot be modeled in a system such as this. One could argue that in a teaching system - they should not be. Part of good teaching is choosing cases that are prototypical or that represent common exceptions. The goal is to help students learn basic classification skills in practice, giving them a framework into which subsequent exceptions can be incorporated.

CONCLUSIONS

We have developed a general set of resources including domain and task ontologies, and abstract problem-solving methods needed for the expert model of an Intelligent Tutoring System in Dermatopathology. The architecture we describe is general, modular and flexible, and could be applied to virtually any feature-based classification task in Medicine. Modifications of this architecture could be used for instruction in other medical tasks such as patient management. The potential reuse of existing resources (e.g. clinical guidelines encoded in GLIF) offers significant advantages to the development of knowledge-based educational systems. By elaborating, expanding and further generalizing this architecture, it may be possible to develop tutoring frameworks (shells) specific to medical tasks. A medical tutoring framework could implement the architecture described, using existing resources like Protégé-2000, to support rapid development of intelligent medical training systems. Use of emergent ideas from the UPML [12] project may one day permit tutoring agents that operate over the distributed knowledge base of the Semantic Web.

ACKNOWLEDGEMENTS

This work was supported by grants from the National Library of Medicine (R01 LM007891-01) and the Competitive Medical Research Fund, Office of Research, Health Sciences, at the University of Pittsburgh.

REFERENCES

1. Crowley RS, Medvedeva O, and Jukic D. SlideTutor – A model-tracing Intelligent Tutoring System for teaching microscopic diagnosis. Proceedings of the 11th International Conference on Artificial Intelligence in Education. Sydney, Australia, 2003.
2. Crowley RS, Naus GJ, Stewart J, and Friedman CP. Development of Visual Diagnostic Expertise in Pathology – An Information Processing Study. J Am Med Inform Assoc 10(1):39-51, 2003.
3. Wenger E. Artificial Intelligence and Tutoring Systems – Computational and Cognitive Approaches to the Communication of Knowledge. Los Altos, CA: Morgan Kaufmann Publishers, 1987.
4. Gennari JH, Tu SW, Rothenfluh TE, Musen MA. Mapping Domains to Methods in Support of Reuse. International Journal of Human-Computer Studies 41:399-424, 1994.
5. Koedinger KR, Anderson JR, Hadley WH and Mark MA. Intelligent tutoring goes to school in the big city. International Journal of Artificial Intelligence in Education 8: 30-43, 1997.
6. Anderson JR, Corbett AT, Koedinger KR, and Pelletier R. Cognitive Tutors: Lessons learned. Journal of the Learning Sciences 4(2): 167-207, 1995.
7. Azevedo R, and Lajoie SP. The cognitive basis for the design of a mammography interpretation tutor. International Journal of Artificial Intelligence in Education. 9:32-44, 1998.
8. Sharples M, Jeffery NP, du Boulay B, Teather BA, Teather D, and du Boulay, G.H. Structured computer-based training in the interpretation of neuroradiological images. International Journal of Medical Informatics: 60: 263-280, 2000.
9. Eliot CR, Williams KA and Woolf BP. An Intelligent Learning Environment for Advanced Cardiac Life Support. Proceedings of the AMIA Annual Fall Symposium, Washington, DC. 1996; 7-11.
10. Clancey WJ. Guidon. J Computer-based Instruction 10:8-14,1983.
11. Anderson JR. Rules of the Mind. Hillsdale, NJ: Lawrence Erlbaum. Associates; 1993.
12. Fensel D, Benjamins VR, Decker S, et al. The Component Model of UPML in a Nutshell. In proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), San Antonio, Texas, 1999.
13. Ackerman AB, Chongchitnant N, Sanchez J, et al. Histopathologic Diagnosis of Inflammatory Skin Diseases. An algorithmic method based on pattern analysis. Second Edition Baltimore, Maryland: Williams and Wilkins; 1997.
14. Motta E and Lu W. A library of components for Classification Problem solving. IBROW Project Deliverable D1–IST-1999. (kmi.open.ac.uk/projects/ibrow/Documents/Class_Libr_Dv1.0.pdf).