PEDAGOGICAL KNOWLEDGE MODEL BASED ON CONCEPTUAL GRAPHS AND ONTOLOGY

Diana STEFANESCU, Emilia PECHEANU, Adina COCU

Department of Computer Science,
University "Dunarea de Jos" of Galati
Str. Domneasca nr. 111, Galati 800201, Romania
phone (+40)/0236/460106
E-mail: Diana.Stefanescu@ugal.ro, Emilia.Pecheanu@.ugal.ro, Adina.Cocu@ugal.ro

Abstract: Intelligent educational systems are knowledge-based systems (KBS) they can be developed by a generic knowledge-based system development methodology. In this paper, we present an ontology-based approach for formalizing different knowledge types. The formalism is based upon conceptual graphs. A priority concern to all research work in adaptive education is that of finding an appropriate representation for pedagogical knowledge. For implementation, we use the CoGITaNT environment (**Co**nceptual **G**raphs **I**ntegrated **T**ools **a**llowing **N**ested **T**yped graphs), a library of C++ classes (open-sources, developed by LIRMM CNRS, France) allowing the development of applications based on the CG knowledge representation scheme.

Keywords: Intelligent educational systems, Pedagogical knowledge modeling, Conceptual graphs.

INTRODUCTION

With the spectacular change that the Web brought to information access worldwide courseware authoring is acquiring a new sense. Web-based authoring courseware could be perceived as a gateway providing personalized access to a variety of Web educational materials.

The new educational systems must integrate artificial intelligence techniques, new technologies (multimedia and Internet), a large array of methods and tools in way that breaks with traditional linear instruction design.

Efficient educational systems should be based on adaptability and reusability. Such intelligent educational systems are knowledge-based systems (KBS) they can be developed by a generic knowledge-based system development methodology.

Explicit representation of *domain knowledge* (in the subject being taught) and *pedagogical knowledge* (how to teach this material) are the two most intensive and complex tasks in building an educational system (Stefanescu et al., 2001).

There is a lack of formalism to express structure, sequencing, presentation and pedagogical uses of the domain content as well as the learning processes involved in it.

In this paper, we present an authoring system and an ontology-based approach for formalizing different knowledge types. The system – still under development - allows to define domain knowledge

This paper was recommended for publication by Viorel MINZU

and pedagogical knowledge base with conceptual graphs.

It should be pointed that the presented system still constitute a rather small steam inside the indeed authoring system. Our intention is to prove that conceptual graphs represent a suitable formalism for constructing domain ontology and pedagogical ontology and for reasoning.

2. CONCEPTUAL GRAPHS BACKGROUND

Conceptual graphs (CGs) is a knowledge representation model (a kind of semantic networks) introduced by J.F. Sowa [1], which uses graphical representation as a method of encoding knowledge and also support computation and automatic reasoning.

2.1. A brief introduction

The CG model is an abstract model which can be used at different levels:

- At a conceptual level, it can be the basis for a specialized communication language between specialists of different domains involved in a common cognitive work;

- At an implementation level, it can be the basis for a common representation tool used by several modules of a complex system, integrating knowledge and databases, inference engines, human-computer interfaces, learning modules, etc.

A CG is a finite, connected (or not), bipartite graph: there are only *two kind of nodes* - *concepts* and *conceptual relations* – and every arc must connect two nodes of different kinds. In a CG, concept nodes are used to represent entities, attributes, states and events, while conceptual relation nodes are used to show how these concepts are related to each other.

CG can be denoted using different representation types:

- Diagrammatic (graphic, display) form – concept nodes are drawn as boxes, relation nodes as circle and arcs as arrowed (or labeled) links connecting these.

- Linear form – a text-based representation, where concept nodes are abbreviated to square brackets, and relation nodes as rounded parenthesis (a no normative representation for human readability; a normative representation - CGIF (CG Interchange Format) – for computer readability).

The CG model is also provided with a formal semantic in mathematical model theory. This is useful to design correct reasoning mechanisms on knowledge expressed.

2.2. The support (the ontology)

Any CG has no meaning in isolation (Chein et al., 1992; Sowa, 2000); a CG is related to a support, which defines syntactic constraints and provides background information on a specific application domain. The support role is to group:

- A set of concept types, representing a AKO (a-kind-of) hierarchy and allowing multiple inheritance. The set of concept types can be (or not) structured in a lattice, with:
  - $\leq$ as order, determined by the *subtype* relation;
  - $\top$ an supremum as universal type;
  - $\bot$ an infimum as absurd type;
  - $\wedge$ as lower bound and $\vee$ as upper bound);

- A set of relation types (structured or not in a lattice);

- A basis, a set of star graphs, showing for every relation type what kind of concept types it can link;

- A set of markers for concept nodes: one generic marker * (for unspecified entities of a given type) and individual markers (to distinguish and name distinct entities, instances);

- A conformity relation, which defines association constraints between a concept type and a marker.

The support provides domain application ontology (domain concepts and domain relation), while a CG represent a proposition (assertion, fact or rule hypothesis or rule conclusion) related to this ontology. Without any predefined concept or relation types, CGs are as ontologically neutral as predicate calculus.

In a CG, the concept node is labeled with:

- The name of concept type

- The referent of concept type:

  - An generic or an individual marker (simple CG);

  - Another CG, named context (nested CG).

In a CG, the relation node is labeled with name of conceptual relation.

A knowledge base is composed of a support (domain ontology), a set of CGs (called facts) and (eventually) a set of rules (also represented by CG hypothesis and CG conclusion).

2.3. Logic interpretation

Sowa proposes (Sowa, 2000) to associate with every CG a well formed formula, based of the Φ operator. The translation of conceptual graphs to first-order logic is done according to certain rules. In logic, the implication operator determines a generalization hierarchy: if a graph or formula *p* implies another graph or formula *q*, then *p* is more specialized and *q* is more general. (It's also possible that *p* and *q* are logically equivalent.).

We found two basic approaches for CG model and reasoning:

- CGs as a graphical representation of logic; reasoning by logical prover (Prolog+CG);
- CGs as a graph-model; reasoning by graph-operation (CoGITaNT).

The second approach is adopted in our work.

2.3. Reasoning in conceptual graph formalism

Since 1991, RCR team (Knowledge and Reasoning Representation team) from LIRMM (Computer Science, Robotic, Microelectronic Montpellier's Laboratory) has been studying CGs as a graphical knowledge representation model, i.e. a model that uses graph-theoretic notions in an essential and nontrivial way.

The aim of knowledge graphical descriptions and reasoning mechanisms consists in providing an interesting alternative to the classical first order logic knowledge representation model.

The key in reasoning with conceptual graph is the operation of specialization/generalization (graph subsumption, equivalent with logic subsumption):

- Let us give the elementary specialization operations and dual elementary generalization operations (both named also formation rules) are internal operations on the set of CG:
  - simplify – addition of twin conceptual relation nodes;
  - restrict – extension of concept nodes or relation nodes
  - join – split.
- The projection operation (a graphs morphism) - a specialization sequence - which permits the design of reasoning mechanisms which are sound and complete with respect to deduction in first order logics (Mugnier et al, 1996) for simple graphs, (Mugnier, 1995) and for nested graphs. A rule application is also based on graph morphism.

The specialization relation is denoted by $\leq$. Let two conceptual graphs (G and H) and G $\leq$ H. In this case, we have:

- G is a specialization of H; H is a generalization of G.
- There exists a projection from H to G.

3. CoGITaNT ENVIRONMENT

For implementation, we use the CoGITaNT environment (open source, developed by RRC – LIRM team), a library of C++ classes allowing the development of applications based on the CG knowledge representation scheme.

This library is not addressed to the end-user, but it can be used for development of conceptual graph based applications. The environment is made as modular as possible: the library module, the server module and the interface module.

CoGITaNT provides un object-oriented model for conceptual graphs (Chein, et al, 1992; Mugnier, et al, 1996; Mugnier, 1995). It provides classes (in the sense of structures + methods) for Conceptual Graph (CG), the main CG operations, the CG forms, Ontology and input/output operations.

Each object has an associated class: `cogitant::Object`, with sub-class `cogitant::Support`, `cogitant::Graph`, `cogitant::ConceptType`, `cogitant::Concept`, `cogitant::Edge`, `cogitant::Operation`, `cogitant::CoreferenceClass`, `cogitant::Rule`.

Class `cogitant::Environment` is the more general class, which groups the support, graphs, rules and operations related to the support.

Main advantages to use CoGITaNT are:

- The client/server architecture, which enables distance-operating based TCP protocol;
- The portability: CoGITaNT can be used under Windows or UNIX operation systems, with different compilers. We have used CoGITaNT under Linux (Cygwin) operating-system and GNU C++ compiler.
- The graphical interface, which enables:
  - Browsing and type hierarchy operations of an ontology created by knowledge engineer;
  - Creation, modification, inspection, verification and errors pointing of CGs;
  - Multiple-views of CG: linear form, graphic form, CGIF, BCGCT (proper intern form), CoGXML (a version of XML);

- Some CG operation.

4. PRESENTATION OF THE SYSTEM

We shall the present the way knowledge representation is implemented and we offer a detailed presentation of specifics concepts. Our work aims at providing an environment (under development) of authoring and presentation of pedagogical multimedia contents adapted to the end-user.

Pedagogical knowledge requires an explicit representation. Our authoring system will consider two knowledge types:

- pedagogical knowledge at macro-level, about course organization (which concept must be teach next in this chapter or lesson?)

- pedagogical knowledge at micro-level, about concept presentation, delivery (example before rigorous definition?)

This paper concerns the second knowledge types. We will use an ontology to describe (figure 1):

- the domain model (all concept from the knowledge domain and relationships between concepts, like: prerequisite, hasPart, etc.).

- the delivery model through pedagogical presentation scenarios.

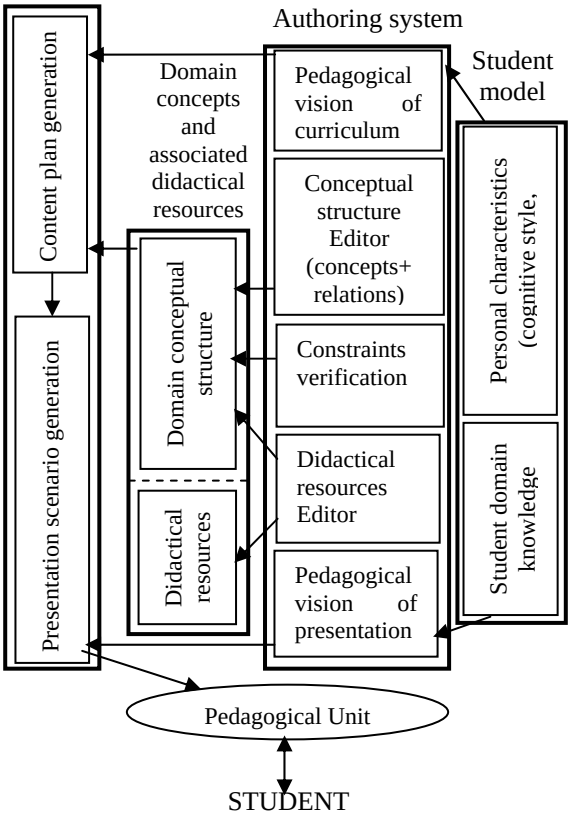

Fig. 1. The role of pedagogical module

4.1. Pedagogical ontology

The knowledge engineer (we) creates the ontology (by programming in C++). The support (the file obtained by executing the program creare_ontologie.cpp) is saved in a file with `.bcs` extension (`ontologie_pedag.bcs`) (as see in following paragraph).

```
//file support ontologie_pedag.bcs
{BCGCT:3;App:"cogitant 5.1.5"}
Begin
   Support: (10,4,0,34);
     TConSet:
       ConceptTypes:
         T;
         ResursaDidactica;
         RolPedagogic;
         Locatie;
         UnitPedagDeInstr;
          Disciplina;
          Capitol;
         Lectie;
         UnitConceptuala;
         UnitPedagCompusa;
       EndConceptTypes;
       Order:
          ResursaDidactica < T;
          RolPedagogic < T;
         Locatie < T;
          UnitPedagDeInstr < T;
         UnitConceptuala <
UnitPedagDeInstr;
          UnitPedagCompusa <
UnitPedagDeInstr;
          Capitol < Disciplina; Lectie <
Capitol;
          Disciplina < UnitPedagCompusa;
       EndOrder;
      EndTConSet;
       TRelSet:
        RelationTypes:
          localizata_in{Signature:2,
ResursaDidactica,Locatie};
           are_rol_pedag{Signature:2,
ResursaDidactica,RolPedagogic};
          refera{Signature:2,
RolPedagogic,UnitConceptuala};
           next{Signature:2,
RolPedagogic,RolPedagogic};
         EndRelationTypes;
           Order:
           EndOrder;
       EndTRelSet;
       TNesSet:
         NestingTypes:
         EndNestingTypes;
         Order:
         EndOrder;
       EndTNesSet;
      Conf:
       Text, ResursaDidactica;
       Tabela, ResursaDidactica;
       Desen, ResursaDidactica;
       Imagine, ResursaDidactica;
       Sunet, ResursaDidactica;
```

```
        Film, ResursaDidactica;
        Simulare, ResursaDidactica;
        Introducere, RolPedagogic;
        Definitie, RolPedagogic;
        Reamintire, RolPedagogic;
        Concluzie, RolPedagogic;
        Demonstratie, RolPedagogic;
        Descriere, RolPedagogic;
        Diferentiere, RolPedagogic;
        Evaluare, RolPedagogic;
        Identificare, RolPedagogic;
        Prezentare, RolPedagogic;
        Recapitulare, RolPedagogic;
        Explicatie, RolPedagogic;
        Intrebare, RolPedagogic;
        Raspuns, RolPedagogic;
        c1, UnitConceptuala;
        c2, UnitConceptuala;
        c3, UnitConceptuala;
        "Programare in C", Disciplina;
        "Limbaje formale", Disciplina;
        "Date, operatori, expresii",
Capitol;
        "Implementarea structurilor de
crontrol", Capitol;
        "Pointeri", Capitol;
       "http\://lib.cs.ugal.ro/~diastef/
curs_an1/def_c1.txt", Locatie;
       "http\://lib.cs.ugal.ro/~diastef/
curs_an1/doc_c1.jpg",
Locatie;"http\://lib.cs.ugal.ro/~diaste
f/curs_an1/prezentare_c1.ppt",
Locatie;"http\://lib.cs.ugal.ro/~diaste
f/curs_an1/concl.doc", Locatie;
       "http\://lib.cs.ugal.ro/~diastef/
curs_an1/rezumat.doc", Locatie;
      EndConf;
     EndSupport;
End.
```

4.2. Pedagogical scenarios based CGs

After, the course author (any teacher) can use this ontology to define the proper pedagogical vision (a pedagogical scenario) about the course (content and presentation). The course author can browse this ontology using the graphical interface and can create, modify and validate the CGs (figure 2).

We have defined in our ontology the concept of PedagogicalUnit, which means an instructional (educational) unit. A pedagogical unit can have different granularity levels: simple (denoted by conceptual unit) or composite (a lesson, a chapter or a discipline) (figure 3). The conceptual unit can have many didactical resources (elementary document fragments) associated; a didactical resource can play a pedagogical role (introduction, definition, example, conclusion, summary, etc.); a didactical recourse have also a certain location (in Internet, Intranet or local computer).

Using the system, the teacher can associate for each conceptual unit didactical resources, can specify for each didactical resource the pedagogical role and the location (figure 2).

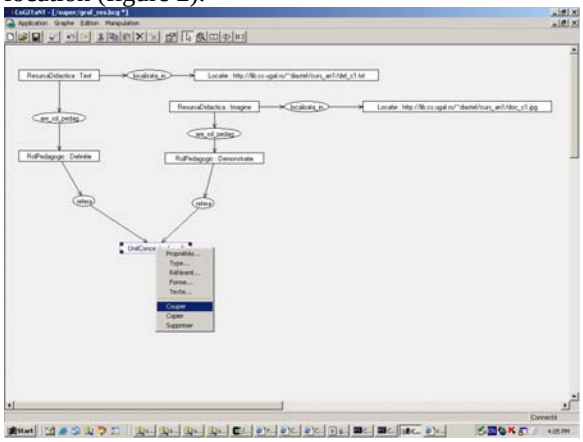

Fig. 2. The graphical interface and the CG graf_res.bcg

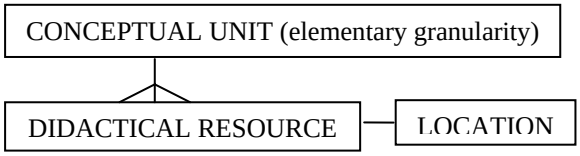

Fig. 3. The entity-relation diagram

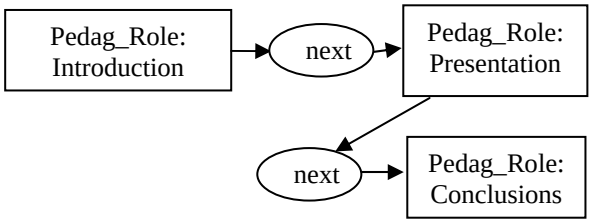

Fig. 4. Pedagogical presentation scenario

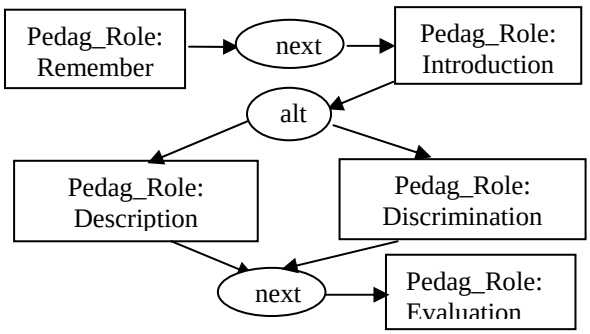

Fig. 5. Pedagogical presentation scenario

The teacher can create (made by graphical interface) many presentation scenarios for a conceptual unit, adapted to the student motivation, student knowledge, student learning style (a definition following by example, following by explanation, fig. 4, 5). The presentation scenario is also represented by CG.

Using the ontology created by the knowledge engineer, the teacher can specify (made graphical interface and CGs created or modified by themselves):

- Which didactical resource will use for a certain conceptual unit;
- Different association between didactical resources and location;
- Different association between didactical resources and pedagogical role;
- Can create different presentation scenarios.

All these are represented by CG and benefit of automatic validation (made same interface).

The system process all knowledge specified by teacher and generates dynamically virtual document (a `.html` page), according with student profile and knowledge.

5. CONCLUSIONS

Conceptual graphs (CG), with their formal structures and operations, appear to be a suitable formalism for constructing domain ontology and pedagogical ontology and for reasoning.

A graph-based reasoning model provides two main advantages:

- From a computational viewpoint, reasoning’s benefit from combinatorial algorithms (from graph theory) and is logically founded;
- From a modeling viewpoint, reasoning’s can be visualized in a natural way and are simple to understand for an end-user. This property is particularly significant for knowledge acquisition.

CoGITaNT tools provides a good start point for developers in knowledge modeling and inference.

The proposed knowledge representation system can produce the models used in a wide range of hypermedia system. Because the dynamic generation of presentation is a separated mechanism from content construction, enhancing presentation reuse and consistency, thus reducing the development cost.

6. REFERENCES

Chein, M. and M.L. Mugnier (1992). Conceptual Graphs: fundamental notions, *Revue d'Intelligence Artificielle*, pp.365-406, vol.6-4.

Mugnier, M.L. (1995). On generalization / specialization for conceptuals graphs, *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 7, pp. 325-344.

Mugnier, M.L. and M. Chein (1996). Représenter des connaissances et raisonner avec des graphes, *Revue d'Intelligence Artificielle*, vol.10-1, pp. 7-56.

Sowa, J.F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole Publishing Co., Pacific Grove, CA, ISBN 0-534-94965-7.

Stefanescu, D., E. Pecheanu and S. Bumbaru (2001). Using pedagogical agents in intelligent tutoring systems, *The 7-th International Symposium on Automatic Control and Computer Science and Parallel Workshop on Control Theory, Modeling, Simulation and Systems' Identification - Programme and abstracts*, SACCS2001, Iasi, CD+ISBN 973-8292-10-7, pp.100.

WWW (2005). http://cogitant.sourceforge.net (accessed on-line)