2015

Diagram-based intelligent tutoring systems

Enruo Guo
Iowa State University

Diagram-based intelligent tutoring systems

by

Enruo Guo

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Computer Science; Human Computer Interaction

Iowa State University

Ames, Iowa

2015

TABLE OF CONTENTS

ACKNOWLEDGMENTS

First and foremost, I would give my deepest thanks to the Holy Father, Holy Son, and Holy Spirit, who directed me to start this Ph.D. program at the darkest time of my life, helped me to find my interests, re-built my confidence, and guarded me all the way in ups and downs. My dearest friend Jesus Christ always gave me wisdom and strength when I was struggling in research and coursework. I can see this is God's miracle that I can finish my Ph.D. in computer science with a co-major in human computer interaction within three years. Praise the Lord!

I would thank my advisor Dr. Stephen Gilbert. I still remember how grateful I was to get the chance to start this project after a talk with him in Skype in 2012. During the study, I am impressed by his enthusiasm in doing research and mentoring students, his frankness, care and sincerity to students, and his efficiency to respond to any kinds of query or request. I feel thankful for his generosity and cultivation to make students become independent researchers. The three-year journey with him is absolutely a joyful and memorable time in my life.

I would thank my co-major advisor Dr. Les Miller, whom I first met when I took a course of him in 2011. When I started my dissertation topic, we always spent two hours to talk every Wednesday afternoon. I was touched by his care and respect to students, in addition to his rigorousness to research. During our meetings, he not only mentored me in research by explaining his hand-written comments on paper, but also gave me lots of insights for future direction through his personal stories. Without his help, I couldn't finish my dissertation on time.

In addition, I would give my special thanks to Dr. Stephen Gilbert and Dr. Les Miller for their time and efforts in revising this dissertation.

I would also give my thanks to the rest of my POS committee: Dr. John Jackman, who was the PI of the problem framer project I worked before, showed his leadership and guidance in helping me finish the two ITSs. Without the financial and technical support from the project and his efforts in developing XDraw, I cannot get my degree in such a short time. Dr. Johnny Wong, whom I met in Ames Chinese evangelical church at the beginning of this program, gave me lots of encouragement on how to grow bigger and stronger in Christ as a professional. I was moved by his role as a faculty but also leads a church with many involvements. Dr. Michael Dorneich, whom I knew during 2013 summer REU program when both of us served as mentors of the undergraduates, inspired me by his patience and love to students, as well as his technique specialty in training systems. Also, I would give a special appreciation to the whole committee. Without all your help, I could not schedule my prelim and final oral exam so smoothly.

I would take this opportunity to thank my Mom and Dad, and all my family members, for their prayers, love, trust and endless supports. I am grateful to have them in my life.

Furthermore, I would like to place my sense of gratitude to Home-for-Heart Chinese Bible study fellowship. I can feel God’s love through these dear brothers and sisters, such as Wei, Cindy, Chunquan, Landy, Jing, Dafang, Tianqi, Chong, Geng, and the rest of the group.

In addition to that, I would thank Pastor David, G.P. and Nancy from Christ Community Church, who gave me lots of inspirations on the walk with Jesus Christ by their life-changing testimonies and teachings. A special thanks to my foreign friends Joori and Daniel, who made my Ph.D. journey more fascinating through many casual conversations, and thanks to be given to other brothers and sisters in the Body of Christ as well.

Furthermore, I would thank all the members in problem framers project: Matt, Gloria, LeAnn and Mostafa, for their two-year collaborations and help to make the project complete. I would give special thanks to my lab-mate Mostafa, who made my Ph.D. study experience enjoyable. Finally, I would thank the ITS team, such as Colin, Jingyu, Jamiahus and Ben for their participation and sharing in our weekly meeting, the ALC group such as Chase, Norene, Curtis and Amanda, and my other colleagues at VRAC.

ABSTRACT

This work first presents two implementations of Intelligent Tutoring Systems (ITSs) on engineering undergraduate-level diagram education: StaticsTutor for free-body diagram and Thermo Cycle Tutor for refrigeration T-v diagram. Initial investigations on several groups of students have shown their educational effectiveness.

Unlike text-based input, diagram has some intrinsic challenges that lead it hard to teach. One example is conceptual knowledge is highly interconnected with procedural knowledge. Learned from the two ITSs, we provided some general pedagogical guidelines for the future Diagram-based ITSs.

Also, we learned classes can be used as a way of representing geometric shapes in diagrams. Thus, we extended our work to the generality of how the current approach can be applied to other domains. We chose a popular type of diagram, called Block Diagram, which contains geometric objects and lines/arrows in connecting them. We developed a methodology to represent a diagram's information and an ontology of diagram evaluation processes to diagnose students' diagrams. Our work contributes to the development of Diagram-based ITSs authoring tools.

CHAPTER 1

INTRODUCTION

In 2014, 34% of all bachelor's degrees were in STEM field according to the report from the National Student Clearinghouse Research Center (2015). Conceptual knowledge learning is one of the crucial parts in STEM education. According to Rittle-Johnson (2006, p. 2), conceptual knowledge is "understanding principles governing a domain and the interrelations between units of knowledge in a domain." Examples include Newton's law in mechanics, heat and laws of thermodynamics. Also, "robust misconceptions" were noticed by researchers regarding force in statics, heat in thermodynamics, etc. They explored reasons why those misconceptions are hard to resolve. One reason might be that students tend to use analogies to help themselves understand those concepts. Thus, it becomes a challenge to teach student conceptual knowledge.

To improve the traditional one-size-fits-all teaching approach, a growing field called "personalized learning" has been investigated in recent two decades. National Academy of Engineering (NAE, 2008) listed "advance personalized learning" as one of the 14 grand challenges for engineering in the 21st century. Intelligent Tutoring Systems (ITSs) are one of the areas to tackle this challenge. ITSs are an application of Artificial Intelligence, which simulate a human tutor by giving tailored feedback to student and present learning contents according to learners' pace and status. Even though ITSs have shown their effectiveness in many fields such as mathematics and physics (Koedinger et al., 1997; Vanlehn et al., 2005), there aren't many systems aimed at engineering education, especially focused on correcting students' misconceptions.

A diagram is a convenient way of conveying conceptual knowledge in many domains. Examples include free-body diagrams in statics, UML modeling diagrams in software engineering, etc. However, unlike text-based representation, a diagram bears some intrinsic challenges from pictorial representation, such as ambiguity and separation of conceptual knowledge from procedural knowledge, which can make it hard to teach and hard to learn. Also, a diagram's underlying representation within a system is quite different from sentential inputs or equations.

To help students convey conceptual knowledge through diagrams, our work contains two parts: 1) By building two Diagram-based ITSs from different domains, we gained hands-on experience on how diagrams can be represented, and explored some pedagogical strategies that are useful for tutoring with diagrams; 2) we constructed a general knowledge representation for diagrams, and described how it can be used to design the evaluation and pedagogy processes used in future Diagram-based ITSs tutors. .

This chapter gives a brief introduction to diagrams and their advantages as a form of knowledge representation. Also, the chapter discusses the intrinsic limits of diagrams and how Intelligent Tutoring Systems (ITSs) can be used to teach them. Furthermore, it describes the gaps between the current Diagram-based ITSs and the generalization to other domains. Finally, the research questions addressed in this work and our contributions to ITSs community are presented. An organization of this whole document ends the chapter.

1.1. The Diagram as an Important Tool in STEM

Human beings use both internal and external representations when solving problems. The internal representations are stored in the brain, whereas the external ones are recorded on paper or another outside medium. External representations include two types of symbolic expressions: 1) Sentential representations, which is usually described by natural language following some sequence; 2) Diagrammatic representation, where the expression corresponds to the components of a diagram (Larkin & Simon, 1987). According to Larkin and Simon (1987), unlike sequential representations, diagrams use location in a 2d plane to index information which is often implicit in sentential descriptions. Therefore, it saves time and cost to use diagrams to make information explicit to use.

As Larkin and Simon (1987) summarized, diagrams have three important features that are superior to sentential descriptions: 1) Locality, which says information in diagrams can by grouped by their spatial location, so that search of relevant information might be avoided in problem-solving; 2) Unnecessary to label, which says labels can be avoided when information can be grouped by its location; 3) Perceptual ease, which says perceptual inferences can be easily obtained by a diagram's intrinsic graphic representation.

As an example, Larkin and Simon (1987) used an energy-level graph (Figure 1.1) in physical chemistry to explain the three features. While this figure is dissimilar from the diagrams that are the focus of this research, it provides a good example for the three features. 1) Locality: Energy is released if electron transition occurs between energy states. The vertical arrows between states show all the possible transitions. The released energy is calculated by the difference of *E1 and E2*, where *E1* and *E2* represent the energy of higher and lower states, respectively. 2) Reduced labeling: The labels in Figure 1.1 are not essential to any of the inferences. 3) Perceptual ease: The diagram shows perceptually the relative size of energy levels, e.g., a greater energy is released by transition_a, as transition_a has a longer arrow than transition_b. However, not all the diagrams carry the three features, e.g., in UML diagrams, the locations and spatial relations among elements are not considered. It is still worthwhile to mention how we can benefit from diagrams. For most cases we discuss in this research, we focus on diagrams that don't describe an actual spatial arrangement, but rather provide cognitive utility and depict relations among non-spatial variables. For example, arrows among objects in UML diagrams reveal some relationship, but the spatial locations of the objects are less important.

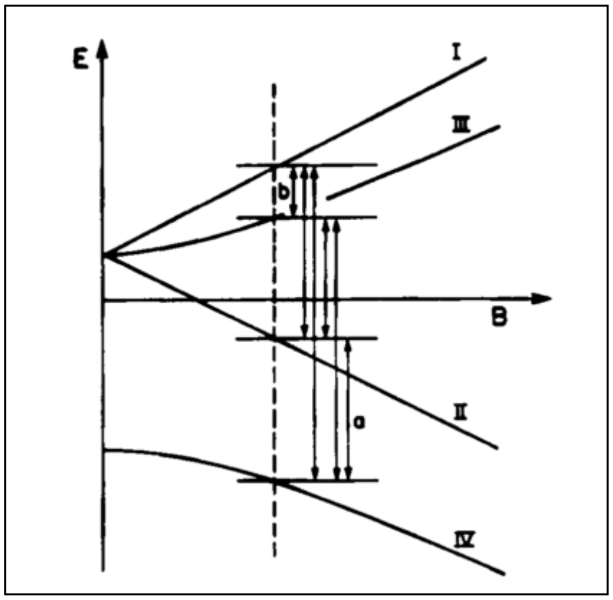


Figure 1.1. An energy level diagram of hydrogen atom. Reprinted from Larkin & Simon (1987).

Based on a series of examples, Larkin and Simon (1987, p. 28) concluded that "diagrams and human visual system provide, at essentially zero cost, all of the inferences we have called 'perceptual'." Therefore, they are used extensively to facilitate communication and comprehension in the scientific engineering world. Also, it is believed that visualized data would not only be understood faster and easier, but also enhance analytical thinking more than corresponding text representations (Shah, 1997).

Given the advantages of pictorial representations, diagrams have been used in many scientific domains at different levels of education. In physics, a free-body diagram is used to represent forces and their relations in a system (Figure 1.2). Forces' relative magnitude and direction are represented. Also, other relevant objects such as the reacting point and contacting plane need to be represented in order to convey the complete message. Furthermore, equations or text descriptions are used to demonstrate student’s knowledge during the problem solving process. Thus, a free-body diagram becomes a major representation of forces and system equilibrium in statics, which is required for most engineering students.

In computer science, Unified Modeling Language (UML) is widely used in software engineering to provide a standard way to visualize the design of a system, which has many applications such as a system diagram, behavior diagram, class diagram, interaction diagram, component diagram, etc. Figure 1.3 shows an example of an UML class diagram. Three classes are included: BankAccount, CheckingAccount, and SavingAccount. Variables and methods are defined within each class. A line with an unfilled arrowhead is used to model the inheritance relationship among the classes. Figure 1.4 shows a class hierarchy of Collection type. The dashed arrow indicates an “implements” relation, whereas the solid arrow indicates an “extends” relation. As we can see, UML diagram and the hierarchy diagram share similar features: they both describe objects and how objects are related. Also, in comparison to Figure 1.1, these diagrams are not spatial. The arrows' direction and connections are important rather than the location of the boxes.

In engineering, a process flow diagram is widely used to demonstrate the general flow of plant processes and equipment, which usually includes major equipment items, process piping, controls and connections, bypass and recirculation stream, names of process stream, and operational conditions such as temperature, density and mass flow rate. An example is shown in Figure 1.5. Even though there are several types of elements, each of them can be abstracted as a block. Connections between the blocks are used to depict the relationship or ownership or successive order between them. Therefore, the process flow diagram has intrinsic similarity to UML and hierarchy diagrams in general.

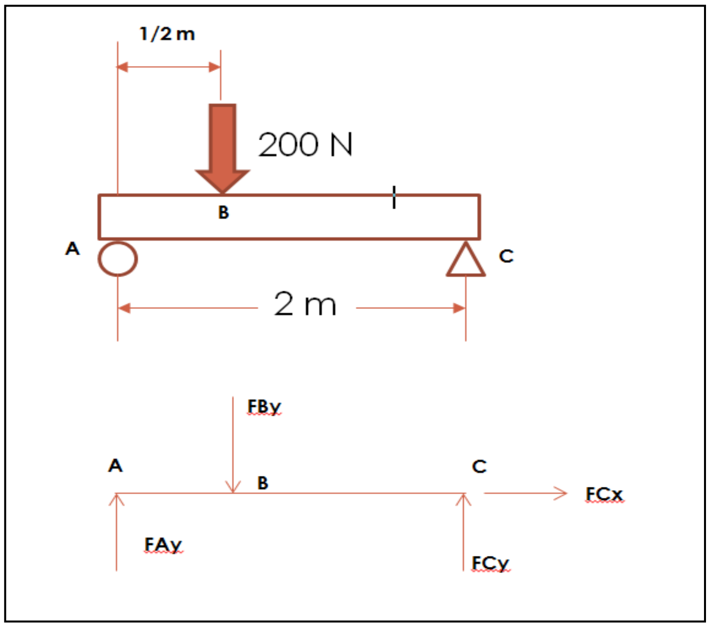


Figure 1.2. A free-body diagram for a beam on a roller and a pin. Used with permission from Gloria Starns.

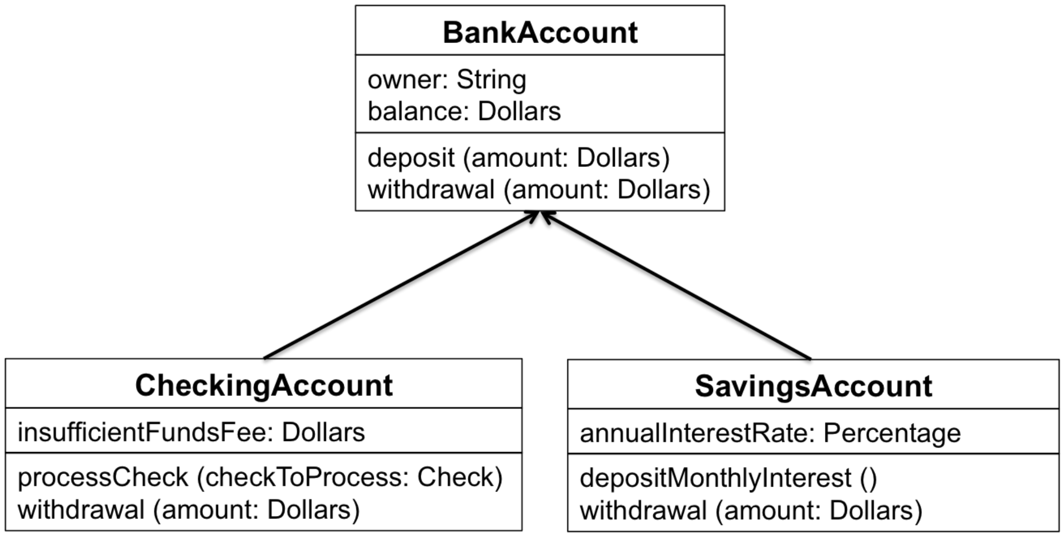

BankAccount
owner: String
balance: Dollars
deposit (amount: Dollars)
withdrawal (amount: Dollars)
CheckingAccount
insufficientFundsFee: Dollars
processCheck (checkToProcess: Check)
withdrawal (amount: Dollars)
SavingsAccount
annualInterestRate: Percentage
depositMonthlyInterest ()
withdrawal (amount: Dollars)

Figure 1.3. An example of UML class diagram. Inheritance is indicated by a solid line with an unfilled arrowhead pointing at the super class.

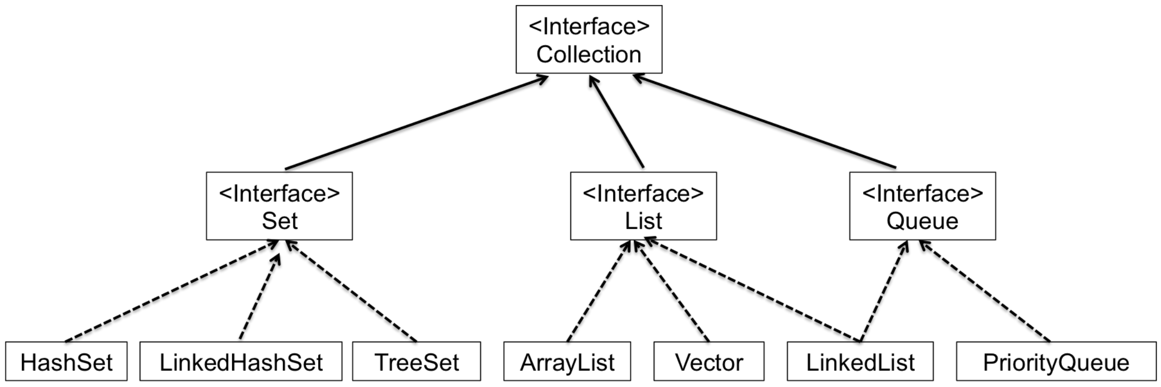


Figure 1.4. A class hierarchy of Collection type. The dashed arrow indicates an “implements” relation, whereas the solid arrow indicates an “extends” relation.

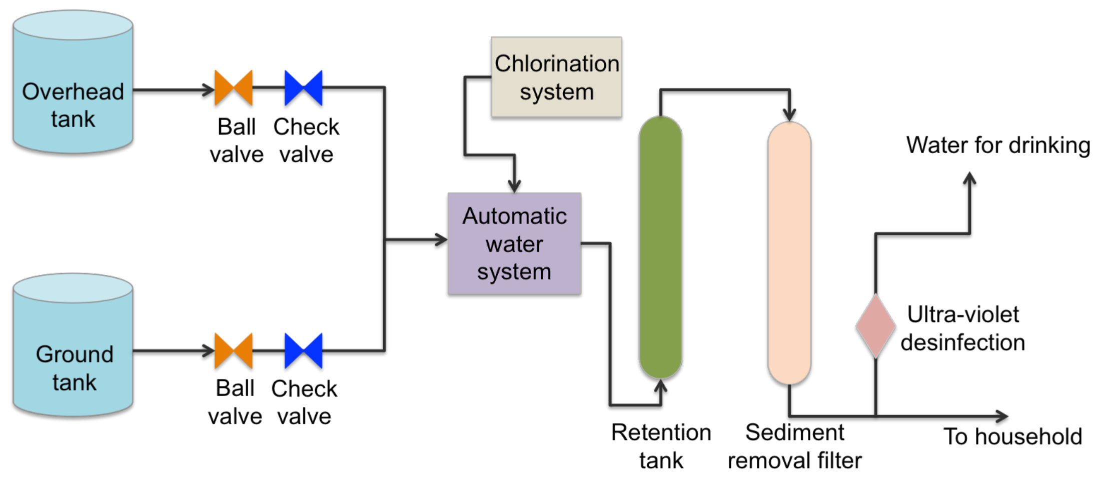


Figure 1.5. A process flow diagram of a drinking water system.

1.2. The Limits of Diagrams as a Way of Communication

Researchers have found graph comprehension is a complex, interactive process (Shah, 1997). Through serial, iterative cycles of identifying and relating the graphic patterns, the viewer would form a mental model of the quantitative information from the graph. Also, both bottom-up perceptual features and top-down factors such as the viewer's expectations or familiarity with the graph content would influence the process. As a diagram is a type of graphic representation, it inherits the intrinsic complexity of graphs. Because of that, in many disciplines, using a diagram to represent knowledge and be understood by others becomes a difficult process. I.e., compared with a text representation, it is harder to convey the message within a strictly defined, domain-dependent diagram. At the same time, instructors have to undergo a more complex process to understand and diagnose a student's diagrammatic representation. Also, many diagrams require specific symbols and predefined format to convey knowledge, which means that students need to learn them ahead of time. Therefore, a big challenge arises:

students have to go through some practice to convey their understanding on a diagram with an appropriate format.

Furthermore, like other communications, diagrams have potential ambiguities. In natural language, many people resolve ambiguity problems based on context, conventions of communication and domain-specific knowledge. However, in diagrams, many ambiguities are subtle and are more difficult to resolve than those in natural language. Even though Futrelle (1999) has suggested some techniques on helping to resolve diagram ambiguity, such as choosing grammar rules to apply in context-specific ways, designing grammars to reflect graphics conventions and examining surrounding text, it is still an open question on how much we can do to reduce diagram's ambiguity.

1.3. The Rise of ITSs and Diagram-based ITSs

Defined by Psotka & Mutter (1988), an Intelligent Tutoring System (ITS) is aimed at giving personalized instructions to learners at an appropriate time. It is designed to simulate a human teacher's behavior and guidance after interpreting student's responses. ITSs have shown their educational effectiveness in many areas. Its one-on-one tutoring paradigm gives students more personalized guidance. ITSs have been used both in class and for homework in mathematics, physics, computer programming, and other subjects (Corbett, Koedinger, & Hadley, 2002; Oliveira Neto & Nascimento, 2012; Rai & Beck, 2012; Sklavakis & Refanidis, 2013; Vanlehn et al., 2005).

To make use of the benefit of ITSs' one-on-one tutoring paradigm, many systems have been developed to teach students how to draw a diagram. A few examples follow. COLLECT-UML (Baghaei & Mitrovic, 2005) is a UML class diagram tutor where students can both learn individually and collaboratively. EER-Tutor (Zakharov, Mitrovic,

& Ohlsson, 2005) helps learning and practicing principles of Enhanced Entity-relationship modeling in database course. Free-Body-Diagram Assistant (Roselli & Howard, 2003) allows students to construct free-body diagrams on a given human body picture and receive constructive feedback in the course of biomechanics. CogSketch (Forbus et al., 2011), which is sketch-based educational software, has demonstrated the powerful ability to understand sketched shapes and recognize them even after rotation or change of position.

In general, like many existing ITSs, a Diagram-based ITS includes: 1) A user interface that usually contains a drawing palette, 2) a pedagogical module which incorporates an expert's feedback and pedagogical strategies, 3) a domain model that contains semantic or syntax constraints or knowledge applicable to all problems in the system, and 4) a student model (Nkambou, 2010; Self, 1990). Based on different ITSs' structures, some of them have a diagnosis module, which compares a student's diagram with expert's solution, if the domain model is unable to automatically generate a solution. If a student's drawing doesn't agree with the ideal drawing, instructional feedback or hints will be given. Because the systems are designed to interpret varied student responses, they are able to determine why a student's understanding has gone astray. Thus, personalized feedback is offered to fix their misconception or misunderstanding at the earliest time possible. As a result, ITSs provide many benefits of a human tutor to very large numbers of students, where the one-on-one interaction promotes students' learning experience. Furthermore, ITSs provide real-time data to instructors and developers to refine their teaching methods in the future.

1.4. Knowledge Representation to Facilitate Authoring Diagram-based ITSs

In the last two decades, some work has been done on developing authoring systems, with the aims at simplifying the process of creating an ITS, and decreasing the skill threshold for the ITS builder. The most complicated aspect of creating an ITS is building the domain model which contains the knowledge for student to learn. In Artificial Intelligence, knowledge representation is the method of encoding knowledge in an intelligent system's knowledge base (Grundspenkis & Anohina-Naumeca, 2015). It basically maps objects and relationships of the real world to computational objects and relationships in a computer-readable format. Thus, constructing knowledge representation about the domain to learn is a desirable feature of ITS authoring systems.

According to Murray (1999), there exists several types of ITS authoring tools. However, very few tools focus on building tutors to teach diagrams. Even though some of the Diagram-based ITSs (Suraweera & Mitrovic, 2004) provide an administration feature which allows instructor to add a new problem and its ideal solution, it is strictly limited to a particular domain. A generalized representation that could capture important features of various types of diagrams in different disciplines would be valuable. Moreover, a structure that explicitly contains an evaluation process and pedagogy instructions would be of great benefit and should simplify the development of Diagram-based ITSs in the future.

1.5. Research Questions

To facilitate the development of Diagram-based ITSs for various types of diagrams, we use classes to represent a diagram's information in the domain model, and an ontological approach to include evaluation steps and pedagogical instructions. An ontology not only benefits knowledge articulation by breaking down a diagram into a set of related elements, but also provides an embedded representation of a diagram, in which authors can directly use such knowledge in diagram diagnosis. Furthermore, an ontology's intrinsic hierarchical structure supports knowledge management and visualization when an author needs to view different types of elements in a diagram. Moreover, re-use of the authored knowledge becomes feasible, since an existing ontology can be easily exported to another ontology-friendly system. Finally, a diagram ontology can support automated procedural knowledge creation. I.e., when the author draws the ideal solution in the system, the drawing order of each element can be recorded and used to create procedural knowledge automatically.

Therefore, the following questions are addressed in this work.

1. What can be learned from creating tutors with different underlying diagram representations?
2. Can we extend this knowledge to other domains? Can a general representation model for diagrams be developed?
3. Can we also develop a methodology to incorporate common evaluation procedures and pedagogy instructions for Diagram-based ITSs?

The first question requires us to develop solutions in some specific domains. The second and third questions ask if we can provide solutions by extending our current methods to more general settings.

1.6. Contributions

This work contains two parts. Firstly, to answer our first question, we implemented two diagram-based ITSs, StaticsTutor for free-body diagrams and Thermo Cycle Tutor for thermodynamics cycle T-v diagrams, based on pedagogical strategies from instructors in undergraduate engineering courses. Initial investigations of several groups of students showed that StaticsTutor can effectively identify students' misconceptions, and that Thermo Cycle Tutor can increase students' learning gains relatively quickly. Even though they belong to different domains, i.e., the T-v diagram is completely different from a free-body diagram, we found there are some similarities when designing the evaluation engine. Both tutors have a drawing palette that allows students to use the basic shapes, such as line, point, rectangle, etc. In the tutor's evaluation engine, each shape has a representation class that contains possible functions of it. One example of such functions is *if_contact (another point)* in class of Line, which checks whether an instance of Line contacts with an instance of Point. The basic idea gained from this work is to use classes to represent properties of the diagram and its components.

Secondly, to answer Questions 2 and 3, we found many existing diagrams belong to the family of Block Diagram, which consists of geometric shapes by using lines or arrows to connect them. Therefore, we chose Block Diagram as our target focus in this work. Chapter 6 not only provides a general class representation of the knowledge in a Block Diagram, but also presents the design of an ontology containing the evaluation process and pedagogy types needed to handle evaluation outputs. This ontology includes three evaluation steps: general property check, block property check and problem-specific check, along with seven pedagogical instruction types. We demonstrate the ability to use this ontology to create knowledge representations and design evaluation processes in three situations: 1) creating a tutor from scratch for a given domain, 2) creating a tutor in a new domain based on the information from an existing domain, and

3) add a new tutored problem in an existing domain. We believe our approach can ease the creation process for future Diagram-based ITSs

1.7. Organization of This Dissertation

In Chapter 2, we offer a literature review on Intelligent Tutoring Systems (ITSs) and their major components, two main methods to model domain knowledge, and some examples of diagram-based ITSs. Also, we include a brief description about desired goals of the ITS authoring process and some existing strategies to achieve those goals. The second part is about ontology, a method of knowledge representation in AI. We discuss uses of ontologies, types, languages and tools to organize them, followed by some evaluation methods. Moreover, a survey of classification of diagrams will be discussed.

Chapters 3 and 5 are extended versions of two conference papers published in the *12th International Conference on Intelligent Tutoring Systems*. Chapter 3 discusses a free-body diagram tutor in an engineering statics course (StaticsTutor), which focuses on providing students tutoring experience at their problem framing stage. Chapter 5 is about pedagogical guidelines and some practical issues for diagram-based ITSs, based on our experience in two ITSs: Thermo Cycle Tutor and StaticsTutor. Chapter 4 presents Thermo Cycle Tutor in more detail, a T-v diagram tutor for an undergraduate thermodynamics course. A detailed system architecture of the tutor, as well as its educational effectiveness is provided. This work will be submitted to the journal *Computers and Education*. In Chapter 6, we describe knowledge representation and ontologies applied to domain modeling in Diagram-based ITSs. Two main ontologies are discussed: a general property ontology for block diagrams and an ontology for evaluation process-pedagogy. Chapter 7 concludes with a summary of the research. Also, future work on another type of diagram is discussed.

REFERENCES

Baghaei, N., & Mitrovic, A. (2005). COLLECT-UML: Supporting Individual and Collaborative Learning of UML Class Diagrams in a Constraint-based Intelligent Tutoring System. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part IV* (pp. 458–464). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/11554028_64

ConceptDraw. (2015). Engineering Diagrams. Retrieved April 3, 2015, from http://conceptdraw.com/samples/engineering-and-technical-drawings

Corbett, A. T., Koedinger, K. R., & Hadley, W. S. (2002). Cognitive Tutors: From the research classroom to all classrooms. In P. S. Goodman (Ed.), *Technology enhanced learning: Opportunities for change* (pp. 235–263). Mahway, NJ: Lawrence Erlbaum Associates.

Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2011). CogSketch: Sketch Understanding for Cognitive Science Research and for Education. *Topics in Cognitive Science*, *3*(4), 648–666. doi:10.1111/j.1756-8765.2011.01149.x

Futrelle, R. P. (1999). Ambiguity in Visual Language Theory and Its Role in Diagram Parsing. In *Proceedings of the IEEE Symposium on Visual Languages* (p. 172–). Washington, DC, USA: IEEE Computer Society. Retrieved April 3, 2015, from http://dl.acm.org/citation.cfm?id=832280.834534

Grundspenkis, J., & Anohina-Naumeca, A. (2015). *Fundamentals of Artificial Intelligence: knowledge representation and networked schemes*. Lecture notes. Retrieved March 3, 2015, from http://stpk.cs.rtu.lv/sites/all/files/stpk/lecture_7.pdf

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent Tutoring Goes To School in the Big City, 30–43.

Larkin, J. H., & Simon, H. A. (1987). Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, *11*(1), 65–100. doi:10.1111/j.1551-6708.1987.tb00863.x

Murray, T. (1999). Authoring Intelligent Tutoring Systems : An Analysis of the State of the Art, 98–129.

NAE. (2008). NAE Grand Challenges. Retrieved April 3, 2015, from http://engineeringchallenges.org/challenges/learning.aspx

NSC Research Center. (2015). National Student Clearinghouse Research Center. Retrieved April 3, 2015, from http://nscresearchcenter.org

Oliveira Neto, J. D. De, & Nascimento, E. V. (2012). Intelligent Tutoring System for Distance Education. *Journal of Information Systems and Technology Management*, *9*(1), 109–122. doi:10.4301/S1807-17752012000100006

ProgramCreek. (2008). The interface and class hierarchy diagram of Java Collections. Retrieved from http://www.programcreek.com/2009/02/the-interface-and-class-hierarchy-for-collections/

Psotka, J., & Mutter, S. A. (1988). *Intelligent Tutoring Systems: Lessons Learned*. Lawrence Erlbaum Associates.

Rai, D., & Beck, J. E. (2012). Math Learning Environment with Game-Like Elements : An Incremental Approach for Enhancing Student Engagement and Learning Effectiveness. *Proceedings of the 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012.*, 90–100.

Rittle-Johnson, B. (2006). Promoting transfer: Effects of self-explanation and direct instruction. *Child Development*, *77*(1), 1–15. doi:10.1111/j.1467-8624.2006.00852.x

Roselli, R. J., & Howard, L. (2003). Integration of an Interactive Free Body Diagram Assistant with a Courseware Authoring Package and an Experimental Learning Management System. In *Proceedings of the American Society for Engineering Education Annual Conference*. Nashville, TN.

Self, J. (1990). Theoretical Foundations for Intelligent Tutoring Systems. *J. Artif. Intell. Educ.*, *1*(4), 3–14. Retrieved from http://dl.acm.org/citation.cfm?id=95885.95888

Shah, P. (1997). A Model of the Cognitive and Perceptual Processes in Graphical Display Comprehension. In M. Anderson (Ed.), *Reasoning with diagrammatic representations* (pp. 94–101). Menlo Park, CA: AAAI Press.

Sklavakis, D., & Refanidis, I. (2013). MATHESIS : An Intelligent Web-Based Algebra Tutoring. *International Journal of Artificial Intelligence in Education*, *22*, 191–218. doi:10.3233/JAI-130036

Suraweera, P., & Mitrovic, A. (2004). An Intelligent Tutoring System for Entity Relationship Modelling. *Artificial Intelligence in Education*, *14*(3-4), 375–417.

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *Int. J. Artif.*

Intell. Ed., *15*(3), 147–204. Retrieved from http://dl.acm.org/citation.cfm?id=1434930.1434932

Zakharov, K., Mitrovic, A., & Ohlsson, S. (2005). Feedback Micro-engineering in EER-Tutor. In *Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning Through Intelligent and Socially Informed Technology* (pp. 718–725). Amsterdam, The Netherlands, The Netherlands: IOS Press. Retrieved from http://dl.acm.org/citation.cfm?id=1562524.1562620

CHAPTER 2

LITERATURE REVIEW

2.1. Intelligent Tutoring Systems (ITSs)

Because this research focuses on using Intelligent Tutoring Systems (ITSs) for education with diagrams, it is worth noting what previous work has been done with ITSs and particularly on Diagram-based ITSs. ITSs are computer-based systems that contain learning contents as well as pedagogy strategies (Wenger, 1987). The earliest version appeared in 1970s by Carbonell (1970), which is called Intelligent Computer-Assisted Learning (ICAI). ICAI usually used hard-coded contents with no dynamic interaction. Later, researchers found that one-on-one tutoring achieved performance two sigma better than traditional teaching (Bloom, 1984), which motivated AI researchers to create intelligent systems that tailored instruction to an individual's needs and ability, benefitting every student. The term "Intelligent Tutoring System" was finally coined by Sleeman and Brown in 1982, in their volume of the same title (1982). As a part of AI in an educational application, an ITS is designed to infer students' mastery of topics based on their performance, so that learning content or style can be adapted dynamically. Also, "mixed-initiative" tutorial interactions are allowed in most ITSs, where students can have more controls over their learning experience by asking questions or requesting hints. In the recent two decades, ITSs have shown its success in many areas (Aleven, Mclaren, Roll, & Koedinger, 2006; A. Corbett, 2001; Koedinger, Anderson, Hadley, & Mark, 1997; Vanlehn et al., 2005).

2.1.1. Four components

According to Self (1990) and Nkambou (2010), an ITS in general contains four components (Figure 2.1).

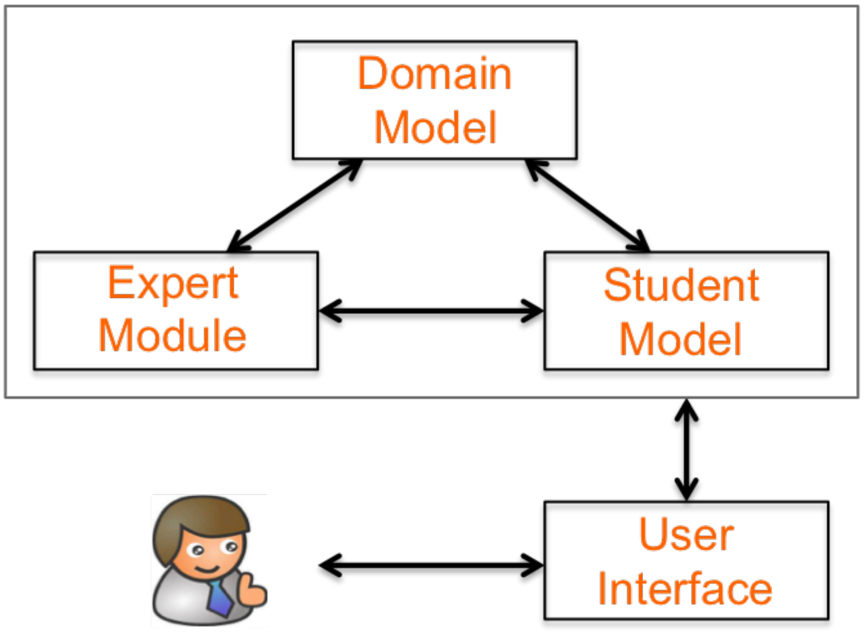


Figure 2.1. The four-component architecture of an Intelligent Tutoring System.

The **Domain Model** contains what the system wants to teach. It could include various types of content in various formats. For instance, a curriculum with a series of knowledge elements could serve as the domain model. The knowledge element can have different granularities, whereas the structure of the curriculum can be organized dynamically such as in hierarchies, semantic networks, etc. As a result, the domain model can serve as a resource of expert knowledge, as well as a standard for diagnosing student's response or a detector for student's errors.

The **Student Model** contains knowledge about the student's cognitive and affective states within the domain, and how they evolve. In a learner-centered ITS, the student model becomes the core component. To achieve this goal, Wenger mentioned in his book

(1987) that three main functions are desired in student model: 1) the student's explicit and inferred data should be acquired; 2) it should contain a representation of the student's state in terms of his/her knowledge and understandings based on the gathered data; 3) it should offer some data-driven diagnosis of what is required to update the student's state of knowledge, selecting appropriate domain materials. Self (1988) identified six roles of the student model: corrective, elaborative, strategic, diagnostic, predictive and evaluative. These goals include help students remove or diagnose bugs in their knowledge, predict their future actions and assess their performance.

The **Expert Module** is where ITSs choose tutoring strategies to help the student during the problem solving process. It uses information from the domain and student models. It involves decisions, which include whether or not to intervene once an error is detected. And if an intervention is needed, it determines when an action should be taken, and how it should be applied.

The student interacts with the tutoring system through a **User Interface**, where different types of interactions might be involved, such as diagram construction, dialog boxes, hints, and evaluation feedback. This component presents domain knowledge through multimedia, simulations, etc.

In general, Diagram-based ITSs have a similar structure as many ITSs. One distinct feature is the user interface, which usually has a drawing tool that allows the user to choose and drag the basic shapes during the problem solving process. However, the major difference is the knowledge representation within the domain model. In the next section, we give a survey of domain modeling in ITSs, and then discuss the methods used in current Diagram-based ITSs.

2.1.2. Modeling domain knowledge

As we mentioned before, domain model contains the learning content to teach, it is critical to have an explicit representation. Also, it is wise to include a problem solver in the domain model, which use the representation for reasoning and getting a solution. Furthermore, the model should be able to interpret student's actions.

Historically, three types of models have been identified: 1) The black box models, where representation is completely inexplicit and only final results are provided. 2) The glass box models, an intermediate model that reasons as human expert, but with different control structure. 3) The cognitive models, which try to mimic human cognition, with respect to representation formalisms and inference mechanisms (Corbett, Koedinger, & Anderson, 1997). Its objective is to support cognitively plausible reasoning. Anderson (1996) proposed a theory, Adaptive Control of Thought (ACT-R), to understand human cognition, upon which several ITSs have been built, such as Algebra Tutor (Singley et al., 1989) and LISP Tutor (Corbett & Anderson, 1992).

2.1.2.1 General purpose languages

According to Nkambou (2010), the selection of a language used to represent domain knowledge must consider four important issues: 1). The expressivity, a measure of the range of constructs, in terms of formality, flexibility, explicitness and accuracy, in describing the components of the domain; 2) The inference capacity, which is based on a formal semantic system and an inference procedure. It requires unambiguity in terms of the language construction; 3) The cognitive plausibility both in terms of language representation and reasoning; and 4) the pedagogical orientation. Also, a compromise has to be made between expressivity and complexity (whether it is computable in real time).

In AI, various languages rooted in logic and mathematics have been proposed to represent knowledge. It usually has an easy-parsed grammar and well-defined semantics. Thus, we will give a brief discussion of such general-purpose languages below.

2.1.2.1.1 Production rules

It was used by many early expert systems, and still is one of the most widely used representation languages, due to its advantages such as natural expression, restricted syntax and sound logic basis. Considering representation structures and reasoning logic, it is cognitively plausible (Anderson, 1982). Many cognitive tutors use production rules to encode the actions of expert and detect the rules the student is violating. Three components are involved in a production rule system: 1) a working memory, which includes all the information it acquires to solve the problem; 2) a rule base, which contains information that can be applied to all the problems within a domain; and 3) an interpreter, which takes in charge of the rule application on selection-execution cycle.

2.1.2.1.2 Semantic network

Semantic network is used to model human memory that has various connections or associations between pieces of information. It contains nodes and links. Nodes are corresponded to physical objects or their abstract representation, whereas links are used to measure the relationship between them. An example is shown in Figure 2.2. Following inheritance and instance links can do basic inference. E.g., Node T (*Tweety*) is an instance of Node B (*Birds*). Other inferences such as path-based or node-based have been discussed by Shapiro (1978).

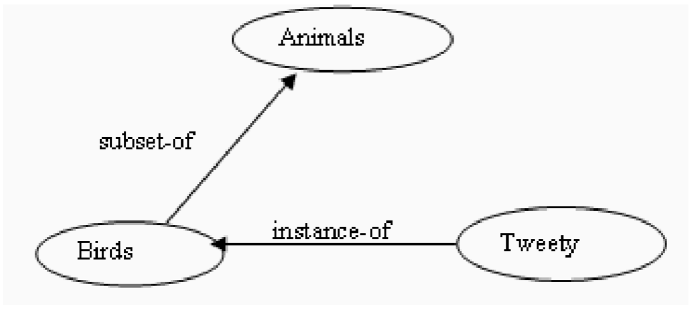


Figure 2.2. A simple example of a Semantic Network, from Nkambou (2010).

2.1.2.1.3 Ontology and description logics

As a formal specification, ontology has a clear definition of concepts and their relationships. It contains two components: a terminology box that describes terminology axioms such as concepts, and an assertion box that includes individuals. It is suitable to build Description Logics, which could provide a rigorous and strong reasoning.

Ontology becomes a great tool to represent propositional knowledge. However, due to its declarative property, it is not expressive enough for description of procedural knowledge. As we know, in addition to conceptual knowledge, learning procedural knowledge in ITSs is also a main focus. So a task ontology is highly desired to specify how problem solving will ideally occur. In other words, the domain modeling should consider the basis of the procedural components, so that the system is more understandable and traceable (Brewster & O'Hara, 2007). Although, there is no standard way of combining procedural knowledge with conceptualized domain knowledge, ease of computation becomes an important factor, which basically needs to make the two levels "loosely coupled" as in ACT-R.

2.1.2.2 Pedagogically-oriented languages

Except for general-purpose languages, there are some other languages that have strong pedagogical functions within the domain knowledge. They are usually in a less formal manner.

Figure 2.3 shows MOT (Paquette, 2008). It contains facts, concepts, procedures and principles as its knowledge units. Each unit is implemented as schema and grammar rules are used to define connections between units. For example, a rule to specialization link shows the transition of all abstract knowledge to other abstract knowledge, either by specialization or generalization means. Domain knowledge formed by links of knowledge units can be exported to OWL (Web Ontology Language).

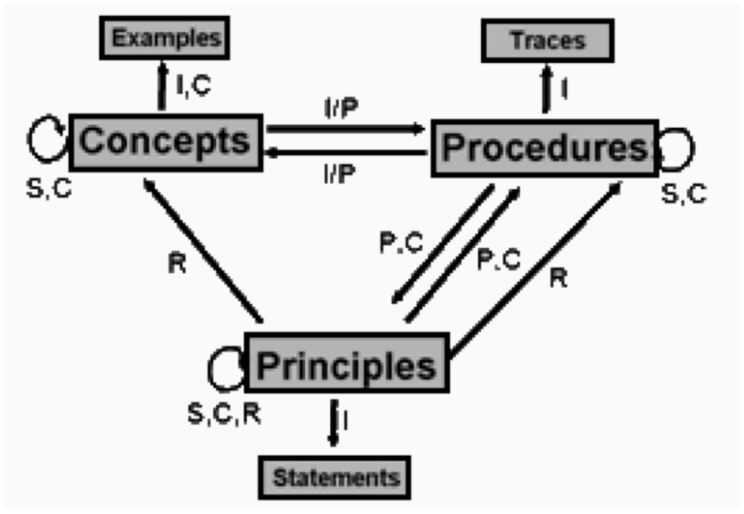


Figure 2.3. Three types of knowledge in MOT, an example of a pedagogical-oriented language, from Paquette (2008).

However, the pedagogical-oriented language has reduced expressivity. It is insufficient to describe complex problems and the unrestrained graphic language cannot guarantee logically sound and semantically valid axioms. Furthermore, the inference procedure is not ensured in terms of completeness and completion within a reasonable time.

In conclusion, general-purpose languages have higher expressivity than pedagogical-oriented languages. In particularly, ontology becomes the choice of knowledge representation in ITS in the era of Semantic Web. It not only supports inferences by providing different levels of expressiveness, but also supports integration of multiple views. Furthermore, it is essentially a declarative approach that creates the domain semantic memory. Thus, it is a solution to capture operational/procedural knowledge, which is on top of the conceptual level. However, without restricted syntax and structures based on a particular domain of diagrams, the knowledge captured by an ontology would be limited. Therefore, other schemes to capture knowledge units can be integrated with an ontology to extend its functionalities.

2.1.2.3 Cognitive modeling in ITS

According to Aleven (2010, p. 1), cognitive modeling "is the activity of producing a detailed and precise description of the knowledge involved in student performance in a given task domain, including strategies, problem-solving principles, and knowledge of how to apply problem-solving principles in the context of specific problems."

Two widely-used models are: 1) rule-based models (Anderson, Corbett, Koedinger, & Pelletier, 1995; Vanlehn et al., 2005) that capture knowledge of generating step-by-step solutions; and 2) constraint-based models (Mitrovic, Mayo, Suraweera, & Martin, 2001) that define requirements applied for all problems in a given domain.

In order to make cognitive model precisely mimics the details of human thinking and problem solving, Aleven (2010) mentioned three concerns when developing cognitive models: 1). Flexibility, which means the model needs to cover wide variety in student's solution path in a given task domain, and different orders of steps in each path. 2). Cognitive fidelity, which requires the model partitions knowledge units or contents in problem solving within the domain based on human's psychological reality. Higher cognitive fidelity would lead to a more precise understanding of the student, and a better-adapted instruction to individual students. It's helpful to have cognitive analysis as part of the model development. 3) Ease of engineering. According to Murray (2003), about 200-300 hours is required to produce 1 hour of tutoring instructions. Lowering the building time is highly desired.

Now we will give a brief discussion about the two types of cognitive modeling.

2.1.2.3.1 Rule-based modeling

Cognitive Tutors, a popular ITS that has shown its effectiveness outside research lab (Anderson et al., 1995; Koedinger et al.,1997), uses rule-based model: production rule. It basically simulates solution paths in which student solves problems. A set of production rules is developed in order to cover possible problem solving paths. By this means, the tutor is able to understand students' performance and track their knowledge that has been defined in the cognitive model. By comparing student's actions in any giving situation with what the model would do in the same situation, the tutor could interpret and access student's solution path. Knowledge tracing algorithm (Corbett & Anderson, 1995), has been used to estimate the probability of student's mastery of each targeted skill.

2.1.2.3.2 Constraint-based modeling (CBM)

Because of the complexity of developing a production set in rule-based modeling and its limitation in capturing all possible mistakes by students through buggy rules, CBM (Ohlsson, 1994) tries to avoid enumerating mistakes. Instead, the fundamentals in CBM are modeling the domain principles. It is based on the assumption that all correct solutions should satisfy the domain principles. CBM uses a set of constraints to capture domain knowledge, where each constraint represents a small section of the domain. Student's solution is thought to be correct only if all the constraints in the given domain are satisfied. Thus, modularity becomes its major advantage.

A constraint contains a pair of relevance condition and satisfaction condition. The relevance condition specifies features of the student's solution that the constraint is important. If it is true, then satisfaction condition should be also true.

Therefore, rather than modeling the origin of a mistake, CBM gives student feedback only when a domain principle is violated. In other words, feedback message is associated with each constraint directly. CBM can accept multiple correct solutions if they do not violate the domain constraints. Furthermore, even solutions that haven't been considered by expert designers can be recognized as correct, at least they satisfy the domain constraints. However, without cognitive modeling, CBM cannot probe the incorrect knowledge that produces the mistake.

2.1.3. Diagram-based ITSs

A few ITSs have been developed to tutor students how to convey knowledge in a diagram. The Andes physics tutor, which is one of the oldest ITSs to teach free-body diagrams, sets a good example on how ITSs can improve students' learning by giving step-by-step feedback. However, it didn't have much emphasis on knowledge representation within its diagrams, as most of the free-body diagram problems contained only forces and two axes. Thus, we choose two other examples below that contain more complicated diagrams, to show how diagram information can be represented in a domain model.

2.1.3.1 CBM in Diagram-based ITSs

CBM is one of the popular and earliest approaches in domain modeling in Diagram-based ITSs. Here we would like to discuss one well-known example: KERMIT (Suraweera & Mitrovic, 2004). It teaches Entity Relationship (ER) Modeling.

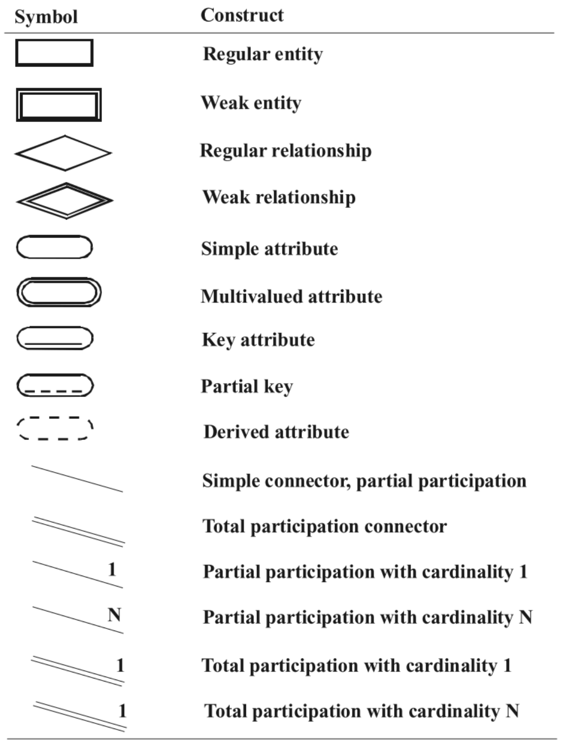
KERMIT supports students problem solving on constructing ER schemas, by providing tailored feedback according to her knowledge. Users can draw ER diagrams by selecting some commonly used objects (shown in Table 2.1) from the UI. The pedagogical module is supposed to present instructional feedback if a certain constraint fails, or select a problem that suits student's current knowledge state. The student modeler which is based on CBM (Ohlsson, 1994), is aimed at comparing student's solution with expert solution and system's knowledge base. The evaluation records will be stored in student model.

As we mentioned the features of CBM, KERMIT diagnoses student's diagram by comparing against ideal solution according to a set of constraints. It accepts both ideal solution same as expert's and the alternative correct ones.

When students start to draw ER diagram, several objects are available in KERMIT to be chosen from. They are shown in Table 2.1. For example, to create an entity node, student needs to select a rectangle or a double rectangle shape, to represent a regular entity or a weak entity. Also, in order to distinguish attribute types visually, they are represented in different shapes.

Microsoft Visio is used as the ER modeling workspace. All the objects are stored in a list according to the order in which they are added. However, objects are in generic type and could not be distinguished by their types in ER diagram. On the other hand, KERMIT dynamically stores all the objects by its own representation. It contains two lists of objects: entity and relationship. A separate list containing attributes is attached to entity or relationship, and relationship list maintains a set of participating entities.

Table 2.1. Symbols available in KERMIT. From Suraweera & Mitrovic (2004).



Symbol	Construct
	Regular entity
	Weak entity
	Regular relationship
	Weak relationship
	Simple attribute
	Multivalued attribute
	Key attribute
	Partial key
	Derived attribute
	Simple connector, partial participation
	Total participation connector
1	Partial participation with cardinality 1
N	Partial participation with cardinality N
1	Total participation with cardinality 1
1	Total participation with cardinality N

Entities = “STUDENT<E1>(Number<K1>), HALL<E2>(Name<K1>)”
Relationships = “LIVES_IN<R1>()<E1>np,<E2>lt-”

Figure 2.4. An internal representation of the expert solution for the scenario “Students live in halls.” Adapted from Suraweera & Mitrovic (2004).

Expert solutions are stored in a textual form (shown in Figure 2.4). The system constructs a runtime representation of the expert solution when a problem is delivered. Objects are grouped by lists, which are similar to student's solution. Diagnosis is based on comparison between the lists of object according to constraint base.

Here we give detailed examples of constraints used by KERMIT, as ER diagrams belong to the subfamily of Block Diagram in which we are interested. We would like to discuss how CBM is applied to model domain knowledge and facilitate diagram evaluation. Figure 2.5 shows an example of a syntactic constraint, Constraint 10. In order to generate feedback, a constraint not only contains relevance condition and satisfaction condition, it also has feedback messages to student when the constraint is violated. The syntactic constraints describe syntactically valid ER schemas.

```
id        = 10
relCond   = "t"
satCond   = "unique (join (SSE, SSR))"
Fedback1  = "Check the names of your entities and relationships. They must be unique."
Feedbck2  = "The name of <viol> is not unique. All entity and relationship names must be
       unique."
Feedback3 = "The names of <viol> are  not  unique. All entity and relationship names must be
       unique."
construct = "entRel"
conceptID = 1
```

Figure 2.5. Constraint 10. A syntactic constraint that specifies all names of entities and relationships need to be unique. "relCond" value is true which means it is always satisfied to all student solutions. "satCond" says that names of all the entities and relations are unique. Three types of feedback messages are attached to compose hints if constraint is violated. "<viol>" tag would be replaced by the name of the constructs that violated the constraint. "Construct" is set to entity and relation. The conceptID is used for problem selection, where the system choses a problem for the student based on her knowledge state. Each constraint is assigned to one concept. From Suraweera & Mitrovic (2004).

Another type of constraints is called semantic constraints. They are used to diagnose student's solution by comparing it with the expert's. An example "Constraint 67" is shown in Figure 2.6, which models the principle that weak entity is equivalent to multivalued attribute. Therefore, KERMIT is able to accept other schema that has the same representation as the correct solution (Figure 2.7).

```
id      = 67
relCond = "each obj ISE
        (and (notNull (matchSS (obj)))
        (and (= type (obj), type (matchSS (obj)))
                  (> (countOfType (attributes (obj), mComp), 0)))))"
satCond = "each obj RELVNT
                  (each att (ofType (attributes (obj)), mComp)
                              (or (and (notNull (matchAtt (att, (matchSS (obj)))))
                                    (= (type (matchAtt (att, (matchSS (obj)))) mComp))
                                 (and (and (notNull (matchSS (att))) (= (type att) w))
                                          (belongs (matchSS (att), obj)))))"
Feedback1 = "Check whether your entities have all the required multivalued composite attributes.
        Your entities are missing some multivalued composite attributes. You can represent composite
        multivalued attributes as weak entities as well."
Feedback2 = "The <viol> entity is missing some composite multivalued attributes. You can
        represent composite multivalued attributes as weak entities as well."
Feedback3 = "<viol> entities are missing some composite multivalued attributes. You can represent
        composite multivalued attributes as weak entities as well."
construct = "ent"
conceptID = 13
```

Figure 2.6. Constraint 67. It deals with composite multivalued constraints, which can be alternatively modeled by weak entity. From Suraweera & Mitrovic (2004).

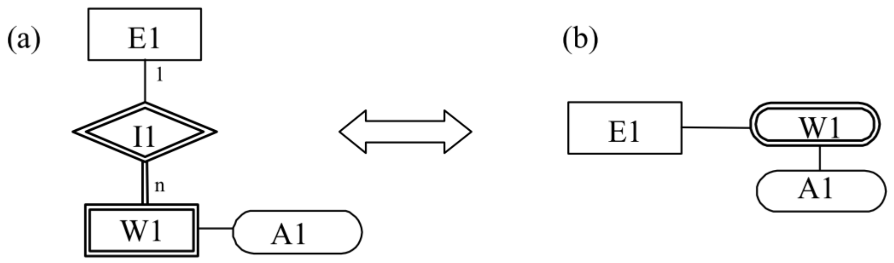


Figure 2.7. The equivalent solution identified by Constraint 67. A weak entity is identical to a multivalued attribute. From Suraweera & Mitrovic (2004).

The CBM has been applied as UML (Unified Modeling Language) class diagram tutor with similar approach (Baghaei & Mitrovic, 2005). As we can see, CBM has shown its success in modeling two types of Block Diagram: ER diagrams and UML class diagrams. However, we find there is still a gap: a generalized methodology of knowledge representation for any type of diagram belonging to the family of Block Diagram. In the next section, we will survey other approaches that have been applied in Diagram-based ITSs, especially the pedagogical-oriented approach by Py et al. (2008), on which our pedagogy ontology discussed in Chapter 6 is based.

2.1.3.2 Other diagram-based ITSs

There are some other ITSs teach diagrams that use less domain modeling. They typically compare student's solution with expert's solution based on some matching criteria, and feedback is enumerated by each condition. Free-body-Diagram Assistant (Roselli, Howard, & Brophy, 2006) is an interactive tool that enables students construct Free-body diagram for human body in biomechanics. The isolated human picture is given when student starts. To interpret which component (force or couple) the student means according to expert solution, three parameters are considered: 1) Type, such as weight, magnetic force or friction force, 2) Location and 3) Orientation. In the first attempt, each component in the student's solution is compared with expert solution. Also, the number of components in student's diagram is checked. Feedback would mention which vectors are correct and whether or not correct number of elements is included. In the second and third attempts, additional information will be provided. For instance, it is able to detect extra elements that are not part of the diagram.

Py (2008) has introduced Diagram. It supports UML diagram modeling. The diagnosis module compares student's diagram with a reference solution based on graph matching algorithm, and produces a list of structure differences between the two diagrams. Structural Differences Taxonomy (SDT, shown in Figure 2.8) is used to label partial matches. Their emphasis is on the design of pedagogical feedback that is built upon SDT (Table 2.2). Feedback has three kinds: 1) Notify, which draws student's attention to some point of the diagram; 2) Questions. By asking binary-choice question, the system encourages the learner to mentally check diagram's property; 3) Suggestions, which contains information about how to correct the diagram. The idea of separating evaluation output from pedagogical feedback inspired us to develop the pedagogy ontology as part of our evaluation process.

Other diagram ITSs are based on sketch or free-hand recognition, such as CogSketch (Forbus et al., 2011), Mechanix (Valentine et al., 2012). These tools give students more freedom to draw shapes they want and label them according to the problem description. However, our work will not focus on these types of ITSs.

Based on the surveys on previous work, we found Py's (2008) work on pedagogical instruction design is very useful, i.e., its approach separates evaluation outputs from pedagogical instructions, which illustrates a method to react to different types of errors appropriately.

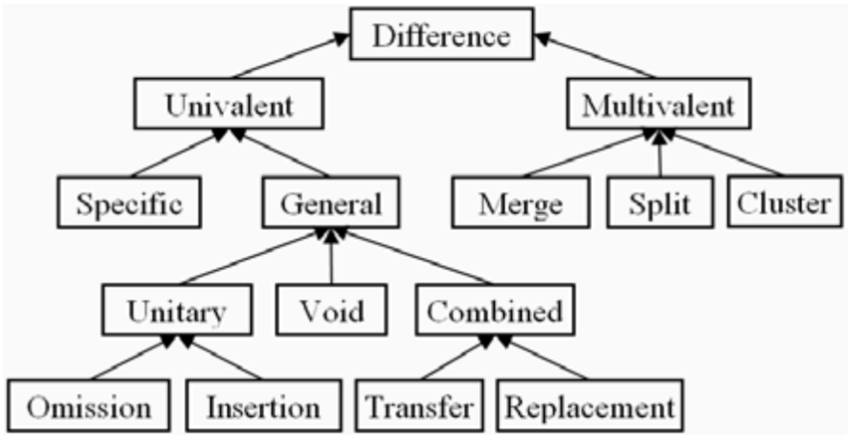


Figure 2.8. Structure differences taxonomy, from Py et al. (2008).

Table 2.2. Correspondence between structural and pedagogical differences, from Py et al. (2008).

Structural differences (SDT)	Pedagogical differences (PDT)
Multivalent - Split	Duplication
Multivalent - Merge	Merging
Multivalent - Cluster	*no match*
Univalent - Specific	Misrepresentation, reversion, bad type…
Univalent - Omission	Omission
Univalent - Insertion	Addition
Univalent - Transfer	Transfer
Univalent - Replacement	*no match*
Univalent - Void	*no match*

2.1.4. Authoring Tools for Intelligent Tutoring Systems

Since ITSs are difficult and expensive to build, authoring systems (or authoring tools) have been developed to support domain experts in creating their ITSs. Some cross-domain tools have been developed, such as GIFT (Sottilare, 2012) and xPST (Blessing, Gilbert, Ourada, & Ritter, 2007; Gilbert, Blessing, & Guo, 2015).

2.1.4.1 Authoring a domain model

As we discussed before, ITSs generally have four components. They are the domain model, expert module, student model and user interface. In this dissertation, our emphasis is on knowledge representation in the domain model. Therefore, we describe several aspects of authoring a domain model (Murray, 1999). Several systems (Munro et al., 1997; Nkambou, Gauthier, & Frasson, 1996) have tools to support visualization of relationships between curriculum elements and authoring content object networks. Some are limited to strict hierarchical representations, but most allow user-defined network representations. Some systems provide WYSIWYG tools. In RIDES (Munro et al., 1997), authors can create components such as switches, levers, pipes, etc. They can also fill in properties for each object such as its state condition, define the connection rules to other components, and constraints between components.

Several types of knowledge can be included in domain expertise. They are problem solving expertise, concepts, facts and procedural skills. Different structures are involved to store different types of expertise. Furthermore, production-rule based modeling or constraints is used to handle more complex skills. However, they are more demanding for non-programmers. Also different knowledge representation schemas are used for different types of knowledge, as different teaching strategies are applied for each type. The DNA system (Shute, Torreano, & Willis, 1998) uses several connections such as *what, how* and *why* to author different types of knowledge and link their relations.

2.1.4.2 Several aspects of building authoring tools

According to Murray (1999), there are five goals in building authoring tools. They are listed below based on their importance:

1. Decrease effort in creating ITS. It includes reduced time, money, or other resources.
2. Decrease skill threshold, which enables more people to participate.
3. Help experts to organize their knowledge.
4. Provide better support for design principles.
5. Enable rapid prototyping.

In order to achieve these goals, several methods were proposed by Murray (1999). We will go through some of them in detail and relate them to Diagram-based ITSs.

1) Using models to scaffold knowledge articulation.

When authoring an ITS, it is a process of knowledge articulation. Therefore, an authoring tool with particular model or framework that scaffolds non-programmers to build tutors is highly desired. One example could be, provide authors with templates to organize and structure their authored information. Also, it is significant to give tools that are able to decompose the contents into a series of knowledge unit, and assist the authors to break down and elaborate the content at each level according to instructional elements and their relations. In a Diagram-based ITS authoring tool, user interface forms with different levels of granularity can be used to articulate information of elements in the diagram.

2) Providing embedded knowledge and default knowledge

Embedded knowledge could be implied as part of the structure in the system, or pre-wired constraints. Or it could be active knowledge that is runnable and produces some results. For instance, 19 types of practice questions are generated as procedural knowledge based on the embedded expertise in XAIDA (Redfield, 1996), where the author can choose when using each type. To author a Diagram-based ITS, default knowledge such as the general properties of the diagram, e.g., connectivity, should be embedded within the authoring tool without requiring additional information.

3) Knowledge management

Authoring systems use many techniques to assist knowledge management and organization, such as use templates, data entry forms or pop-up menus to support input. If the input is limited to a finite set of value, a “choose” mode should be enabled instead of asking author to type. Also, the design of authoring tool should try to separate the representation of content and tutoring strategy. However, due to the complexity and diversity of different types of information in ITS, models are interconnected and cannot be made completely independent. Furthermore, to assist manipulation of large information spaces, tools that allow author to visualize both details and abstract representation are highly desired. This property of an authoring tool is especially important when authoring a diagram. E.g., an expert frequently needs to view the detailed information from a part of a diagram simultaneously with the whole structure.

4) Knowledge visualization

Visualization tools are significant to help authors comprehend large amount of interconnected information. However, the level of interactivity of the visualization tools is primitive, comparing to many off-the-shelf software. A good example is LAT (Sparks, Dooley, Meiskey, & Blumenthal, 1999). It has tools to visualize conversational grammars, which consists of individual scripts that define the possible actions and decision points in the conversation. Moreover, it supports visualization of both static structure and run-time dynamics. In Diagram-based ITSs, for example, visualization tools could support class authoring for each drawing object by displaying the number of terminals in each object, its relative location, and how it relates to properties in class representation.

5) Knowledge re-use

Many pieces such as domain knowledge, pedagogical strategies and user interface, could be re-used. To author an ITS, the relative information can be imported from the library and be re-used by any system. Many benefits are included, such as reduce author's workload and maintenance effort, and increase consistency. But it would also increase the cost in developing the library, and make the system harder to use. One aspect of knowledge re-use with respect to Diagram-based ITSs is the re-use of the basic objects from a drawing tool across many problems. Each object, when re-used, brings associated properties and knowledge with it to a new diagram tutor.

6) Automated knowledge creation

This is one of the most interesting features, which can save author from entering, deriving and articulating information. Authoring system could infer or produce knowledge based on author's demonstration or interaction with the system. One example is RIDES (Munro et al., 1997). By recording expert's action it is able to create operational procedure knowledge. In Diagram-based ITSs, this feature is desirable, as procedural knowledge about diagram construction can be automatically generated if the system can capture the sequence of how an expert draws a diagram by adding each additional piece.

2.2. Knowledge Representation and Ontology

In this section, we will give a short survey on knowledge representation, followed by a thorough discussion on ontology. As we mentioned before, an ontology could be a good solution in domain modeling in ITSs. Here, we offer a more detailed overview of ontology design, components, types, usage and evaluation.

2.2.1. Knowledge representation in AI

In Artificial Intelligence, knowledge representation is the method to encode knowledge in an intelligent system's knowledge base. It maps objects and relationships of the real world to computational objects and relationships in computer-tractable form. In general, knowledge contains five categories (Grundspenkis & Anohina-Naumeca, 2015). 1) Declarative knowledge describes what is known about a problem. It usually lists statements about concepts or facts. 2) Procedural knowledge provides directions on how a problem should be solved. It usually contains rules or strategies. 3) Heuristic knowledge, also called shallow knowledge, describes a rule-of-thumb to guide the reasoning process, and usually compiled by an expert through the experience of solving past problems. 4) Meta-knowledge aims at enhancing problem solving efficiency by referring to the other types of knowledge. It is used to pick other knowledge that is best suited for problem solving. 5) Structural knowledge describes an expert's overall mental model of the problem. It contains rule sets, concept relationships, concept to object relationships and so on. A diagram contains procedural knowledge about the order of pieces appearing in the diagram and the role of each piece. Also it contains structural knowledge about the relationships among the pieces.

Knowledge representation is a set of conventions about how to describe the knowledge. The goal of this representation scheme is not only to capture the essential features of a problem domain, but also to support knowledge application in a problem solving process. In practices, four types of knowledge representation schemes have been studied.

1) Logical scheme, which uses mathematical or orthographic symbols and inference rules. It is strictly based on syntax and semantics.
2) Procedural scheme, which contains a set of instructions for problem solving. It usually uses "IF-Then" rules to capture procedure knowledge. It is allowed to separate a knowledge base from an inference mechanism.

3) Networked scheme, where knowledge is represented as a graph. Nodes of a graph display objects or concepts in a domain, and arcs define relationships between objects and their attributes. An ontology is a type of networked scheme, which we will discuss in the next a few sections.
4) Structured scheme, which extends network scheme by displaying each node as a complex data structure.

Of these four, the most relevant knowledge representation in diagrams is the networked scheme, where graphs can be used to model relations among objects in the diagram.

2.2.2. Ontology definition

The term "ontology" originates in philosophy. Initially, it refers to the study of being or existence and the organization of reality (Guarino & Giaretta, 1995; Studer, Benjamins, & Fensel, 1998). Ontology can be understood as a manner of organizing related concepts under a common specification, in order to facilitate knowledge sharing.

According to two widely adopted definitions provided by Tom Gruber (1993) and Willem Borst (1997), Jakus et al. (2013, p. 29) defined ontology as following:

"An ontology is a formal and explicit specification of a shared conceptualization."

In general, it is considered as a network with additional semantic relations. It uses nodes as concepts, connecting by arrows to represent their relationship. Knowledge sharing is one of the key roles of ontologies in computer science. Ontology contains a few key features (Studer et al., 1998):

a. Ontologies are conceptualizations. They have abstract models that include concepts to describe the real world, which act as a sort of surrogate of reality.

b. Ontologies are explicit. Concepts, relations and other components are defined explicitly.

c. Ontologies use formal language since they are intended to be readable by the computers.

d. Ontologies are shared by a group of users who commit to a set of terms and consensual knowledge.

Based on these features, an ontology is potentially a reasonable representation for a pedagogical process because it can describe the knowledge units explicitly using formal language that can be shared by the community of educators.

2.2.3. Ontology components

Before start to think about constructing an ontology, some issues need to be addressed. Firstly, one needs to select relevant components from the domain model in order to adequately describe a domain. Secondly, one has to consider the level of its generality and reusability. Finally, ontology can be constructed by different approaches: manually, semi-automatically or completely automatically.

According to varies resources (Gomez-Perez & Corcho, 2002; Khoo & Na, 2006; Studer et al., 1998), the component of ontologies generally include: a) concepts, classes, collections, set or types; b) objects, individuals, instances or entities; c) attributes, properties, or features of concepts or objects; d) attribute values; e) relations among classes or/and objects. Ontology is typically built on top of a taxonomy, which is a hierarchical structure of concepts with limited "is-a-kind-of" relation among them. Ontology then expands to a richer network by adding semantic relations and additional components such as functions, restrictions or constraints. Figure 2.9 is an example of ontology with its taxonomical framework highlighted. This framework is further enriched by other semantic relations represented with dashed arrows and italic text.

By decomposing diagram information into smaller components, such as objects, attributes and relations, we can potentially use an ontology, along with classes, as an appropriate representation for diagram knowledge. In Chapter 6 we explore this idea further.

2.2.4. Ontology design

According to Noy & McGuinness (2001), there are some fundamental rules:

1). No single "ideal" solution. Alternatives always exist. Criteria including potential applications, extensibility and maintainability could apply to choose the appropriate one.

2). It requires iterative process. By applying it to real-world problems or by discussing it with the domain experts, we can evaluate and revise our initial design, which continues through the entire lifecycle.

3). As the abstract concepts model the real objects, they need to maintain objects' relationships in the domain of interest.

Four main stages are involved in the ontology lifecycle: 1) Specification, where the purpose and scope will be defined. I.e., applications that might use the ontology, and the competency questions it should answer; 2) Formalization, where models are constructed to meet the requirement from specification stage; 3) Maintenance, which not only tracks the ontology's changes and evolution, but also detects if any inconsistency occurs; and 4) Evaluation, which checks if the existing ontology meets the requirements (Guarino & Welty, 2002).

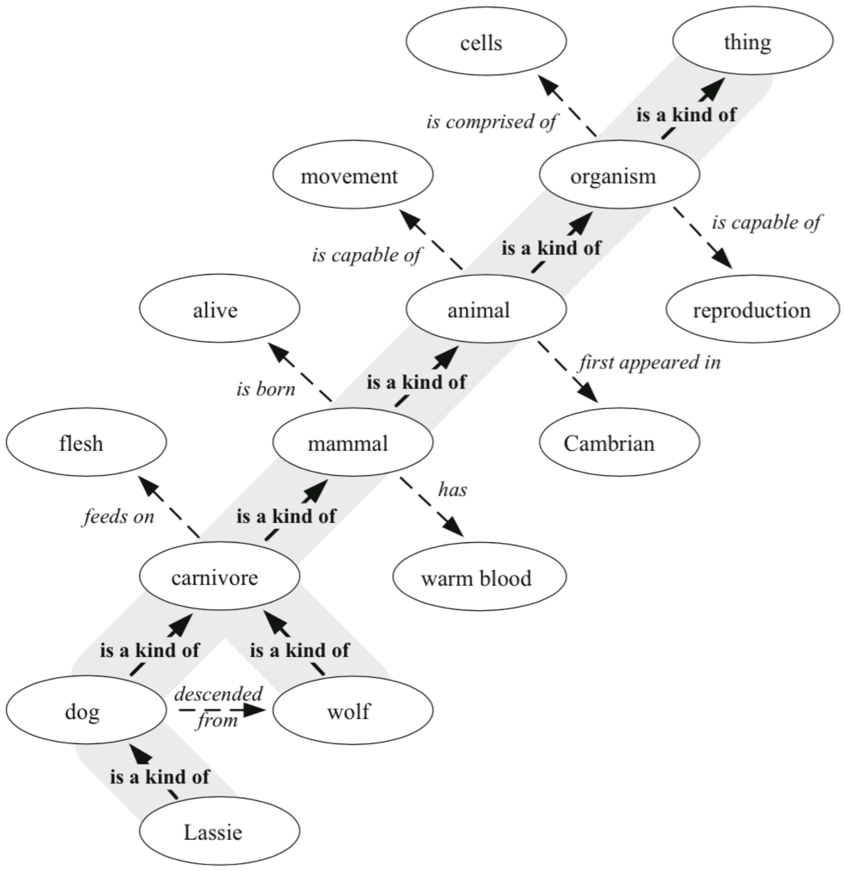


Figure 2.9. An example of ontology with a highlighted taxonomical structure. Picture is reprinted from Jakus et al. (2013).

Therefore, there isn't a unique solution for modeling diagram knowledge and its evaluation processes. For our research, the solution depends its potential applications and the extensibility to other domains. Also, as an initial investigation, we only focus on the first two stages in the ontology lifecycle.

2.2.5. Types of ontologies

As we mentioned in previous content, ontologies are a sort of conceptualizations of reality. To make it effectively reusable and to avoid developing new ontologies when they are already available, it is necessary to divide ontologies into two levels of generality.

a. Generic ontologies, which are also called general, top-level or core ontologies, contain knowledge that can be reusable across various domains. Very general and domain-independent concepts are included in this type, such as events, time, space, matter, actions, causality and behavior. Examples of generic ontologies are Cyc (Lenat & Guha, 1989; OpenCy, 2012), Dublin Core (DCMI, 2015), and Suggested Upper Merged Ontology (SUMO) (2015).

b. Specific ontologies, on the other hand, contain concepts that are specific to a particular domain, application, task, method, etc. More than one type of specific ontologies can be involved to address a particular problem. For instance, "ontology-driven information systems" proposed by Guarino (1998) suggested, in addition to top-level ontology, three distinct specific types of ontology can be involved: domain ontology, task ontology and application ontology (Figure 2.10). Domain and task ontologies contain terms representing concepts of a particular domain and a task or an activity, respectively. Concepts in both types are specialized from a top-level ontology. Application ontology contains subsets of concepts from domain and task ontologies that can be used in a particular application.

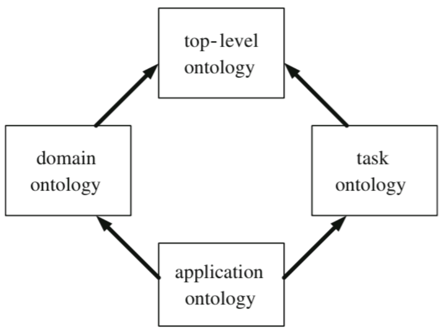


Figure 2.10. Types of ontologies in an ontology-driven information system. Reprinted from Guarino (1998).

In our research, we are interested in specific ontologies, i.e., a domain ontology and a task ontology, which can be used to model diagram knowledge and its related tasks. An application ontology may be useful for modeling different types of diagrams.

2.2.6. Languages and tools to organize ontologies

Dedicated formal languages are used to encode ontologies, which allows the sharing of knowledge and support machine processing by providing reasoning rules. In general, ontology language can be classified into two groups.

a). Description logic which was originated from the field of artificial intelligence. Two typical examples of such languages are KL-ONE (Brachman & Schmolze, 1985) and LOOM (MacGregor & Bates, 1987).

b). First-order logic includes CYCL (Lenat & Guha, 1989), KIF (Genesereth & Fikes, 1992), Ontolingua (Gruber, 1993) and Common Logic (2007).

With the advent of WWW, a new family of ontology languages was created with the intent of knowledge sharing on the Internet. Examples include Simple HTML Ontology Extension (SHOE) (Heflin, Hendler, & Luke, 1999), XML-based Ontology exchange Language (XOL) (Karp, Chaudhri, & Thomere, 1999), and Web Ontology Language (OWL) (2012). In addition, ontologies can also be represented by conceptual graphs and semantic networks.

There are some well-known software tools to create ontologies manually, such as TERMINAE (Biebow & Szulman, 1999; TERMINAE, 2012), Protégé (Noy et al., 2001; Protege, 2014), OntoStudio (Semafora, 2012; Weiten, 2009), and HOZO (Mizoguchi, Sunagawa, Kozaki, & Kitamura, 2007). Besides that, approaches of creating ontologies automatically or semi-automatically from existing information resources such as text document (Fortuna, Grobelnik, & Mladenic, 2007), web document (Navigli & Velardi, 2004) and multimedia (Jaimes & Smith, 2003) have been developed, which is called "ontology learning".

However, as an initial conceptual design, we are not focused on implementing the ontologies using formal languages. Also, the existing software for ontology creation doesn't provide a good visualization tool to support our applications. Therefore, we constructed the ontologies manually, which gives us more flexibility and a better visualization.

2.2.7. Ontology use

The primary purpose of the effort to define and organize ontologies is to facilitate knowledge sharing. It has been used in numerous cases, such as intelligent agent, semantic web services, natural language processing, media content annotation and knowledge extraction. For example, intelligent agent in pervasive computing environments, where agents cooperate with each other and with other devices and services in order to support human activities and needs in an "anywhere, anytime fashion" (Chen, Finin, & Joshi, 2003). One example is the Intelligent Context Broker (ICB). ICB integrates information from different sources and combines into a coherent model. The model is eventually used for reasoning and shared with other agents.

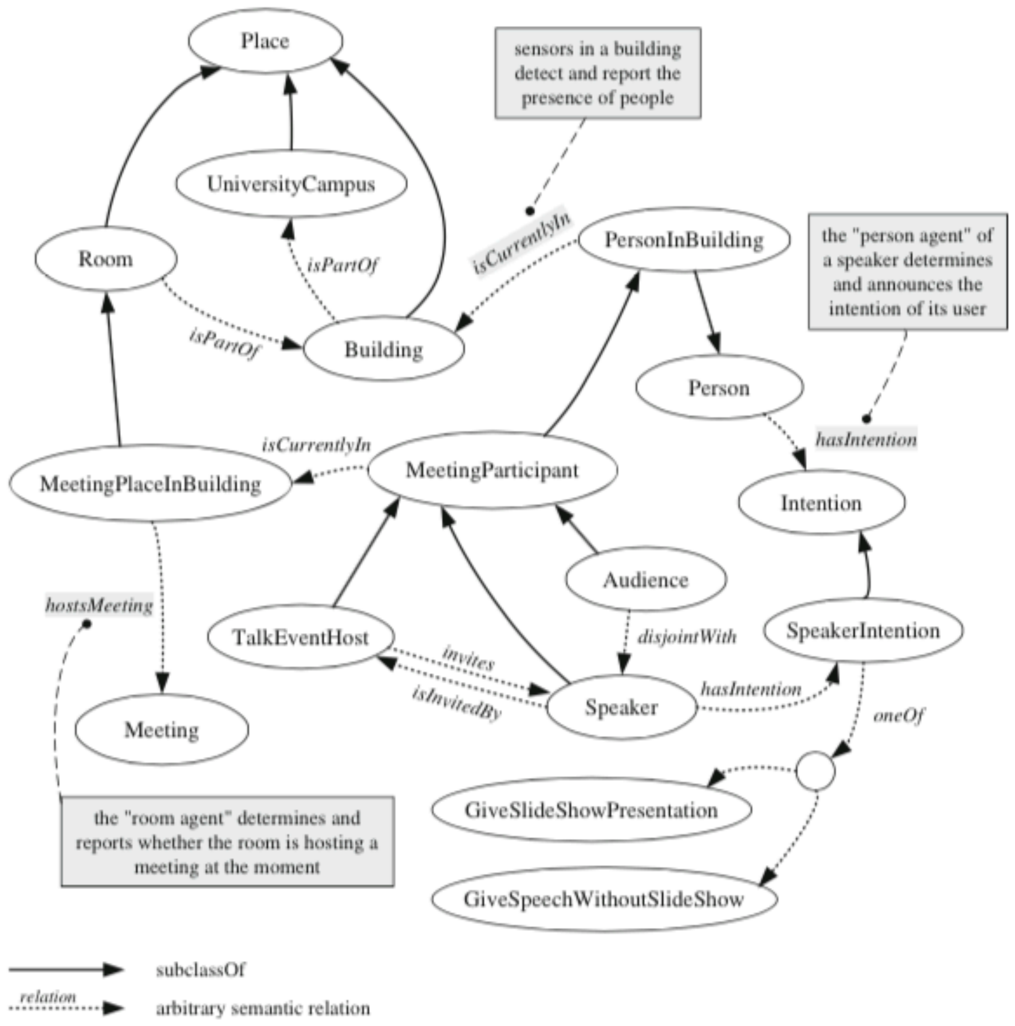

Place
UniversityCampus
sensors in a building detect and report the presence of people
Room
isPartOf
isCurrentlyIn
PersonInBuilding
the "person agent" of a speaker determines and announces the intention of its user
isPartOf
Building
Person
isCurrentlyIn
hasIntention
MeetingPlaceInBuilding
MeetingParticipant
Intention
Audience
hostsMeeting
disjointWith
TalkEventHost
SpeakerIntention
invites
isInvitedBy
Speaker
hasIntention
oneOf
Meeting
GiveSlideShowPresentation
the "room agent" determines and reports whether the room is hosting a meeting at the moment
GiveSpeechWithoutSlideShow
subclassOf
relation
arbitrary semantic relation

Figure 2.11. A part of the ontology supporting the intelligent context broker, from Chen et al. (2003).

Figure 2.11 shows part of an ontology supporting the task ICB. The ontology is used to model a situation in which a speaker gives an invited speech or a presentation at a meeting in a particular room in a building. The room also hosts other people as the audience. Environmental agents are included too, which provide environmental information, such as if a particular room is hosting an event, what are the intentions of the participants and their representation group.

Ikeda, Seta, & Mizoguchi (1997) discussed an ontology-based authoring tool for computer-based training (CBT) system. The tool allows the user to design the skeleton of the CBT, such as its process of interest. Three layers of ontology is defined: a) core task ontology with the least dependency lays foundation for other layers by defining inherent concepts; b) task-type specific ontology, on the second level, is a theory of concepts/vocabulary to describe task models of a certain type of task; c) task-domain ontology, on the top level, describes domain models from the task-type perspective.

The ontology from ICB provides a good example from which to learn within the context of our own research. Unlike the domain ontology, it is more dynamic, involving events, people and other resources for reasoning. Thus, we consider our work can incorporate both a domain ontology and task ontology to fulfill the goal of modeling both a diagram's knowledge representation and its evaluation process.

2.2.8. Ontology evaluation

Once ontology is built, an evaluation is expected which guarantees the initial requirements are met. Evaluation is an ongoing process throughout the lifecycle of management and activities. Also, ontology evaluation can be applied to test the homogeneity of some automatically-generated ontologies from different resources, where duplicate instances or clusters might exist. In addition, it is useful for end users to select the best ontology from several ontologies with similar areas of interest. Generally speaking, current approaches include three dimensions.

a. Evolution-based

As ontology changes over time, this approach tracks changes across different versions. It not only provides the quality of the ontology, but also detects or recovers any invalid changes during the evolution. It could happen due to the changes in the domain, conceptualization or the explicit specification (Noy & Klein 2004).

b. Logical (or rule-based)

Logical approaches use rules to detect conflicts in ontologies. The rules can be built-in rules in the ontology languages, or user-defined. For instance, if two classes are defined as disjoint relation, the ontology cannot have statements that an instance is a member of both classes. This technique seeks to evaluate the quality of ontology and detect any possible errors.

c. Metric-based (or feature-based)

By gathering different types of information presented in the ontology, metric-based approaches evaluate ontology quality from a quantitative perspective, oQual (Gangemi, Catenacci, Ciaramita, & Lehmann, 2006) evaluates ontologies in three dimensions: 1) structural, 2) functional, and 3) usability profiling. OntoClean (Guarino & Welty, 2004) uses a set of four features (Rigidity, Identity, Unity and Dependence) to evaluate each class in the ontology, and further identifies any problematic areas. This approach would result in classes being moved up or down in the ontology hierarchy, or new classes being added.

However, in our current work, the above-mentioned ontology evaluation methods are not appropriate, as the ontologies we developed are small in size, with limited instances. Also, due to the limited applications, there are no significant evolutional changes.

2.3. Diagrams

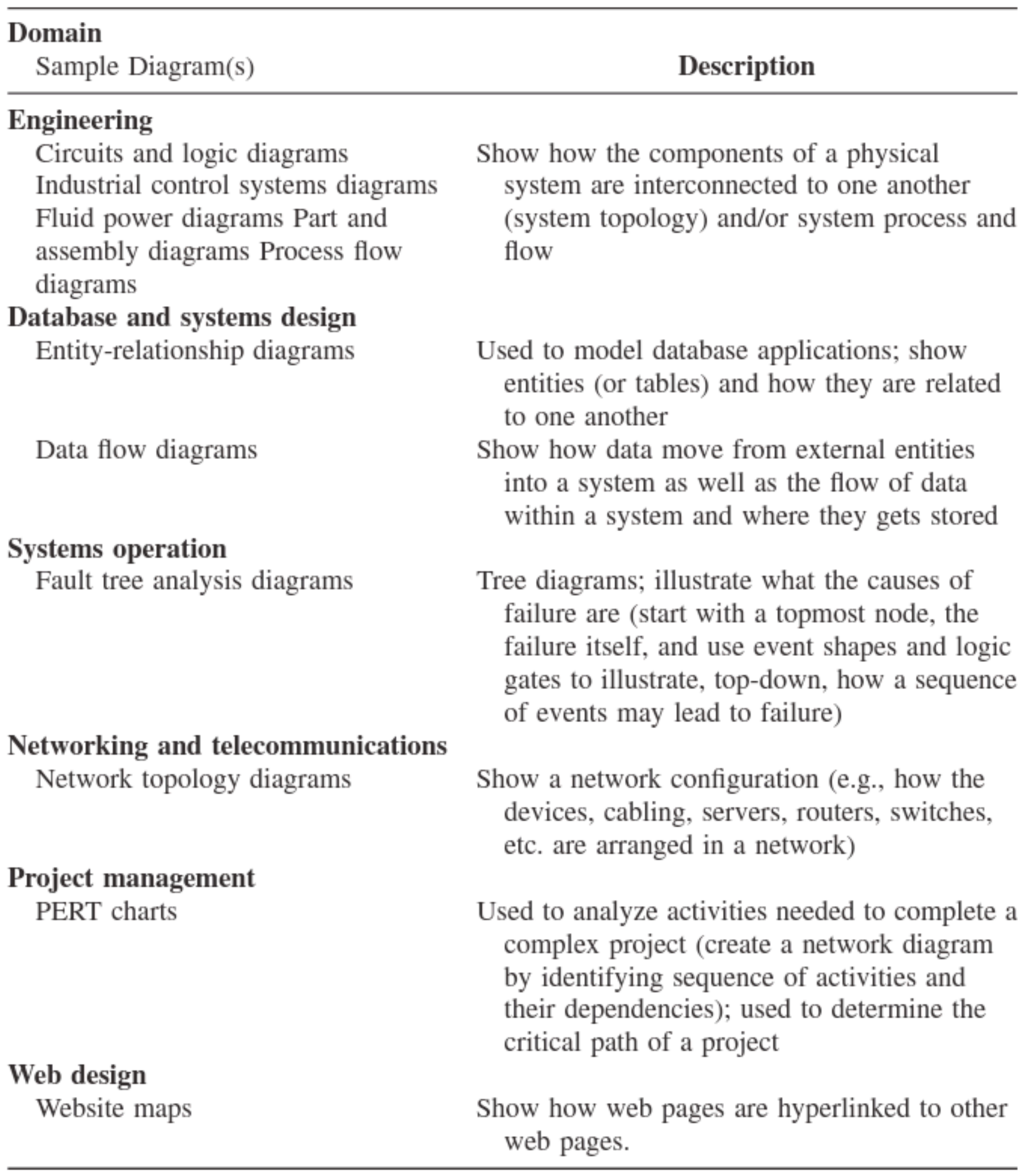
A diagram is a type of information graphic that “preserves explicitly the information about topological and geometric relations among the components of the problem.” (Larkin & Simon, 1987, p. 2) Information is indexed by its location. It facilitates grouping related pieces of information, which supports perceptual inferences.

However, information graph has broader scope than diagram. According to Nakatsu (2010), there are four main types: 1) charts and graphs with quantitative values; 2) maps which shows spatial and directional relationship; 3) tables which includes rows and columns to demonstrate some meanings; and 4) diagrams. There are some other types that do not fit into any of these four categories, such as pictorial illustrations.

Nakatsu (2010, p. 58) gave the definition of diagram as “information graphics that are made up primarily of geometric shapes, such as rectangles, circles, diamonds, or triangles, that are typically (but not always) interconnected by lines or arrows. One of the major purposes of a diagram is to show how things, people, ideas, activities, etc. interrelate and interconnect.” There are a few advantages diagrams possess over verbal descriptions. 1) Information representation becomes more efficient by stripping down information complexity to its core. 2) Diagrams can illustrate patterns in data that may appear disordered in text. 3) Diagrams can reduce ambiguity with a more structured description.

There are enormous varieties of types of diagrams. Table 2.3 shows a few samples of diagrams in some domain areas. Examples include, circuit or logic diagrams used by electrical engineers; Entity-relationship diagram that is used by database developer to serve as a blueprint of future relational tables; and website maps that show its structure.

Table 2.3. Some domain-specific diagramming techniques. Reprinted from Nakatsu (2010).



Domain Sample Diagram(s)	**Description**
Engineering	
Circuits and logic diagrams Industrial control systems diagrams Fluid power diagrams Part and assembly diagrams Process flow diagrams	Show how the components of a physical system are interconnected to one another (system topology) and/or system process and flow
Database and systems design	
Entity-relationship diagrams	Used to model database applications; show entities (or tables) and how they are related to one another
Data flow diagrams	Show how data move from external entities into a system as well as the flow of data within a system and where they gets stored
Systems operation	
Fault tree analysis diagrams	Tree diagrams; illustrate what the causes of failure are (start with a topmost node, the failure itself, and use event shapes and logic gates to illustrate, top-down, how a sequence of events may lead to failure)
Networking and telecommunications	
Network topology diagrams	Show a network configuration (e.g., how the devices, cabling, servers, routers, switches, etc. are arranged in a network)
Project management	
PERT charts	Used to analyze activities needed to complete a complex project (create a network diagram by identifying sequence of activities and their dependencies); used to determine the critical path of a project
Web design	
Website maps	Show how web pages are hyperlinked to other web pages.

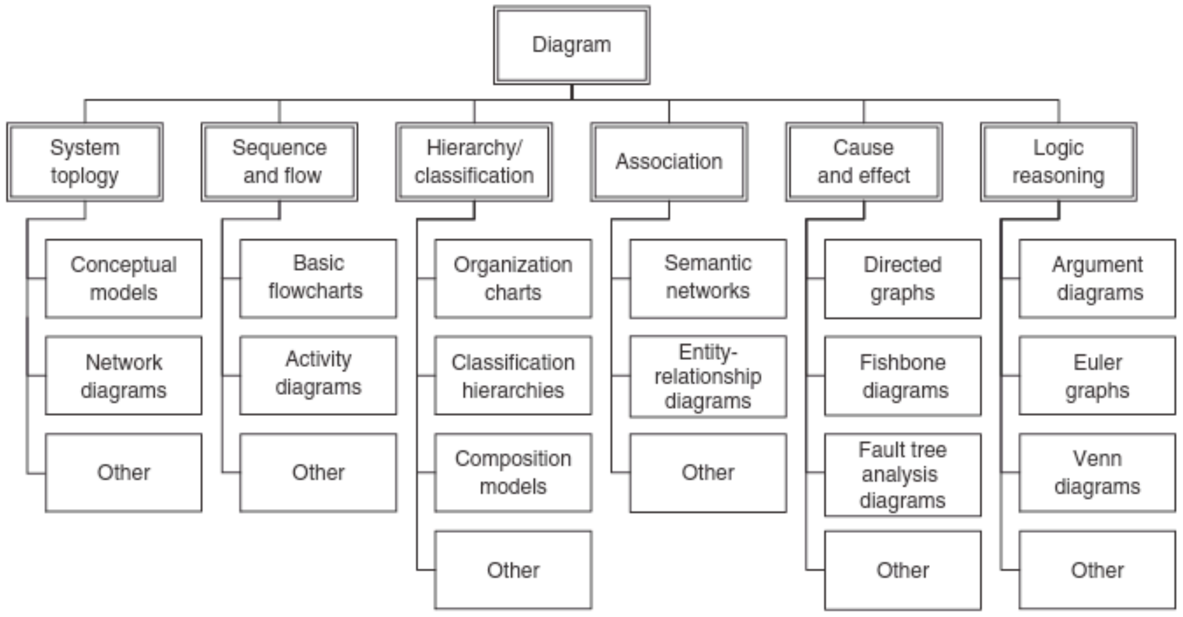


Figure 2.12. A classification of diagrams. Reprinted from Nakatsu (2010).

2.3.1. Classification of diagrams

A classification of diagrams are provided by Nakatsu (2010), shown in Figure 2.12. In general, it is categorized into six subtypes: a) System topology; b) Sequence and flow; c) Hierarchy and classification; d) Association; e) Cause and effect and f) Logic reasoning. We will describe the first five categories in more detail below, since the last category, Logic reasoning, uses a different representation scheme for the purposes of inferences and logic reasoning tasks.

2.3.1.1 System topology

This type of diagram usually uses a conceptual model to represent the organization of components. An example of this kind is a network diagram, containing icons to represent different kinds of components (PCs, printers, servers, etc.) in the network. Also, abstract shapes such as circles, rectangles, triangles, etc. can be applied.

2.3.1.2 Sequence and flow

A flowchart is the best example of this type of diagram. Flowcharts are very common and typically show temporal or chronological orders. They contain sequenced activities and decision nodes. A data flow diagram is another type of flowchart, which shows how data flows into or out of a system, or where and when it shall be stored.

In general, three basic shapes are used. 1) An ellipse represents the terminator, which begins and ends a flowchart. This shape sometimes is left out. 2) A rectangle represents the action or process. 3) A diamond represents a decision point. A question needs to appear at the entry point in the diamond node, where different answers would result in different paths. In general, arrows are used to connect all shapes, with an indication of the sequence of activities.

2.3.1.3 Hierarchy and classification

Hierarchy is a diagram showing parent-child relationships. Three uses of hierarchy were discussed by Nakatsu (2010) : 1) an organizational chart to represent organizational structure, 2) an inheritance hierarchy to represent taxonomy or classification, and 3) composition models showing the whole-to-part relationships.

2.3.1.4 Association

An association diagram generally models the relationship between objects, e.g., in a semantic network or entity-relationship diagram. In AI, a semantic network is a knowledge representation scheme that supports efficient information retrieval. Basically, three parts are involved:

Unit: words or phrases that represent the object.

Property: words or phrases that represent the feature or attribute of the units.

Pointer: unidirectional or bidirectional association between the units.

An example of semantic network might illustrate the relationships of characters in a movie, with the rectangles representing the main characters and the links denoting their relationships. In software engineering, an entity-relationship diagram (ER diagram) is used to describe the data or information aspects of a business domain or its process requirements, which will later be implemented in a database. Although there are varieties of ER diagram, the most general components are:

Entity, which is represented by a rectangle, is the subject the system tries to model. In the future, it should be converted to a database table.

Relationship, which is represented by diamond, shows how entities are related.

Cardinality denotes entity occurrence. It usually has values like one-to-one, one-to-many or many-to-many.

2.3.1.5 Causality

A causal diagram is a diagram that depicts causal relationships between objects. Arrows are used to link the objects, with the cause node pointing to the effect node. Examples include directed graphs with all links directed and fault tress to show events that will cause a fault or an undesirable event.

2.4. Conclusion

In this chapter, in order to help answer our three research questions, we gave a survey on Intelligent Tutoring Systems (ITSs), knowledge representation in AI and diagrams classification.

To help answer our first research question, which requires us to implement real ITSs to teach diagrams, we described the four components in ITSs, and a few domain modeling approaches. We discussed two Diagram-based ITSs, KERMIT and Diagram, which used constraint-based modeling and structure difference taxonomy, respectively. We learned that knowledge representation in diagrams can be broken down into small objects from a drawing palette. Also, it is wiser to separate evaluation outcomes from the pedagogical feedback. We adapted this idea when designing diagram evaluation ontology by adding additional pedagogical-oriented nodes as a separate part (Research Question 3). Then we mentioned a few suggestions for creating future ITSs more efficiently, such as providing embedded knowledge, knowledge re-use, and knowledge management. These suggestions become guidelines when we want to design a general diagram knowledge representation in domain modeling, which relate to our second research question.

To help answer Research Questions 2 and 3, we then discussed ontology and knowledge representation in AI. The purpose of this part is to provide a basic understanding of ontology types, components and designs. To make the survey complete, we also discussed ontology evaluation, even though it is not our focus. Furthermore, a discussion on diagrams was provided, which is important in categorizing diagrams and getting common attributes from each type. This part plays an essential role in helping develop the general property ontology in Chapter 6.

REFERENCES

Aleven, V. (2010). Rule-Based Cognitive Modeling for Intelligent Tutoring Systems. In R. Nkambou, J. Bourdeau, & R. Mizoguchi (Eds.), *Advances in Intelligent Tutoring Systems* (pp. 33–62). Springer Berlin Heidelberg.

Aleven, V., Mclaren, B., Roll, I., & Koedinger, K. (2006). Toward meta-cognitive tutoring: A model of help seeking with a cognitive tutor. *International Journal of Artificial Intelligence in Education*, *16*(2), 101–128. doi:10.1.1.121.9138

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, *98*(4), 369–406.

Anderson, J. R. (1996). *The architecture of cognition*. Lawrence Erlbaum Associates Inc., NJ.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences*, *4*(2), 167–207.

Baghaei, N., & Mitrovic, A. (2005). COLLECT-UML: Supporting Individual and Collaborative Learning of UML Class Diagrams in a Constraint-based Intelligent Tutoring System. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part IV* (pp. 458–464). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/11554028_64

Biebow, B., & Szulman, S. (1999). TERMINAE: A Linguistic-Based Tool for the Building of a Domain Ontology. In *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management* (pp. 49–66). London, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=645360.650759

Blessing, S., Gilbert, S., Ourada, S., & Ritter, S. (2007). Lowering the Bar for Creating Model-Tracing Intelligent Tutoring Systems. *Artificial Intelligence in Education*, *158*, 443–450.

Bloom, B. (1984). The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring, *Educational Researcher*, 13:6(4-16).

Borst, W. (1997). *Construction of engineering ontologies for knowledge sharing and reuse*. Centre for Telematics and Information Technology.

Brachman, R. J., & Schmolze, J. G. (1985). An overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, *9*(2), 171–216. doi:http://dx.doi.org/10.1016/S0364-0213(85)80014-8

Brewster, C., & O'Hara, K. (2007). Knowledge representation with ontologies: Present challenges—Future possibilities. *International Journal of Human-Computer Studies*, *65*(7), 563–568. doi:10.1016/j.ijhcs.2007.04.003

Carbonell, J. R. (1970). AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *Man-Machine Systems, IEEE Transactions on*, *11*(4), 190–202. doi:10.1109/TMMS.1970.299942

Chen, H., Finin, T., & Joshi, A. (2003). An Ontology for Context-aware Pervasive Computing Environments. *Knowl. Eng. Rev.*, *18*(3), 197–207. doi:DOI:10.1017/S0269888904000025

CommonLogic. (2007). Common Logic Working Group. Retrieved February 16, 2015, from http://cl.tamu.edu

Corbett, A. (2001). Cognitive Computer Tutors : Solving the Two-Sigma Problem, 137–147.

Corbett, A. T., & Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, *4*(4), 253–278.

Corbett, A. T., Koedinger, K. R., & Anderson, J. R. (1997). Inteligent Tutoring Systems. In T. K. Helander & P. Landauer (Eds.), *Handbook of Human-Computer Interaction*. Elsevier Science, Amsterdam.

Corbett, A.T., Anderson, J. R. (1992). The LISP intelligent tutoring system: Research in skill ac-quisition. In J. Larkin, R. Chabay, & C. Scheftic (Eds.), *Computer Assisted Instruc-tion and Intelligent Tutoring Systems: Establishing Communication and Collaboration*. Erl- baum, Hillsdale.

DCMI. (2015). Dublin Core Metadata Initiative Home Page. Retrieved February 16, 2015, from http://dublincore.org

Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2011). CogSketch: Sketch Understanding for Cognitive Science Research and for Education. *Topics in Cognitive Science*, *3*(4), 648–666. doi:10.1111/j.1756-8765.2011.01149.x

Fortuna, B., Grobelnik, M., & Mladenic, D. (2007). OntoGen: Semi-automatic Ontology Editor. In *Proceedings of the 2007 Conference on Human Interface: Part II* (pp. 309–318). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=1766591.1766627

Gangemi, A., Catenacci, C., Ciaramita, M., & Lehmann, J. (2006). Ontology evaluation and validation. *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, *3*, 140–154.

Genesereth, M., & Fikes, R. (1992). *Knowledge interchange format, version 3.0, reference manual*.

Gilbert, S., Blessing, S., & Guo, E. (2015). Authoring effective embedded tutors: An overview of the Extensible Problem Specific Tutor (xPST) System. *International Journal of Artificial Intelligence in Education.* DOI: 10.1007/s40593-015-0045-0.

Gomez-Perez, A., & Corcho, O. (2002). Ontology languages for the semantic web. *IEEE Intell Syst*, *17*(1), 54–60.

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowl Acquis*, *5*(2), 199–220.

Grundspenkis, J., & Anohina-Naumeca, A. (2015). *Fundamentals of Artificial Intelligence: knowledge representation and networked schemes*. Lecture notes. Retrieved March 3, 2015, from http://stpk.cs.rtu.lv/sites/all/files/stpk/lecture_7.pdf

Guarino, N. (1998). Formal ontology and information systems. In *Proceedings of formal ontology in information systems*. Trento, Italy.

Guarino, N., & Giaretta, P. (1995). Ontologies and knowledge bases, towards a terminological clarification. In M. NJI (Ed.), *Towards very large knowledge bases* (pp. 25–32).

Guarino, N., & Welty, C. (2002). Evaluating Ontological Decisions with OntoClean. *Commun. ACM*, *45*(2), 61–65. doi:10.1145/503124.503150

Guarino, N., & Welty, C. (2004). An overview of ontoclean. In S. Staab & R. Studer (Eds.), *Handbook on ontologie* (pp. 151–159). Springer Berlin Heidelberg.

Heflin, J., Hendler, J., & Luke, S. (1999). *SHOE: a knowledge representation language for internet applications*.

HOZO. (2012). Hozo: ontology editor. Retrieved February 16, 2015, from http://www.hozo.jp

Ikeda, M., Seta, K., & Mizoguchi, R. (1997). Task ontology makes it easier to use authoring tools. *IJCAI International Joint Conference on Artificial Intelligence*, *1*, 342–347.

Jaimes, A., & Smith, J. R. (2003). Semi-automatic, data-driven construction of multimedia ontologies. In *Proceedings of 2003 International Conference on Multimedia and Expo. ICME '03.* Baltimore. doi:10.1109/ICME.2003.1221034

Jakus, G., Milutinovic, V., Omerovic, S., & Tomazic, S. (2013). *Concepts, Ontologies, and Knowledge representation*. springer.

Karp, R., Chaudhri, V., & Thomere, J. (1999). XOL: An XML-based ontology exchange language, version 0.4. Retrieved February 16, 2015, from http://www.ai.sri.com/pkarp/xol/xol.html

Khoo, C. S. G., & Na, J.-C. (2006). Semantic relations in information science. *Annual Review of Information Science and Technology*, *40*(1), 157–228. doi:10.1002/aris.1440400112

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent Tutoring Goes To School in the Big City. *International Journal of Artificial Intelligence in Education,* 8, 30–43.

Larkin, J. H., & Simon, H. A. (1987). Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, *11*(1), 65–100. doi:10.1111/j.1551-6708.1987.tb00863.x

Lenat, D. B., & Guha, R. V. (1989). *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

López, M. F., Gómez-Pérez, A., Sierra, J. P., & Sierra, A. P. (1999). Building a Chemical Ontology Using Methontology and the Ontology Design Environment. *IEEE Intelligent Systems*, *14*(1), 37–46. doi:10.1109/5254.747904

MacGregor, R., & Bates, R. (1987). *The loom knowledge representation language*.

Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001). Constraint-Based Tutors: A Success Story. In *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Engineering of Intelligent Systems* (pp. 931–940). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=646863.707788

Mizoguchi, R., Sunagawa, E., Kozaki, K., & Kitamura, Y. (2007). The Model of Roles Within an Ontology Development Tool: Hozo. *Appl. Ontol.*, *2*(2), 159–179. Retrieved from http://dl.acm.org/citation.cfm?id=1412401.1412406

Munro, A., Johnson, M. C., Pizzini, Q. A., Surmon, D. S., Towne, D. M., & Wogulis, J. L. (1997). Authoring Simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*, *8*, 284–316.

Murray, T. (1999). Authoring Intelligent Tutoring Systems : An Analysis of the State of the Art, 98–129.

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In S. Murray, T., Blessing, S., Ainsworth (Ed.), *Authoring tools for advanced learning environments*. Kluwer Academic Publishers, Dordrecht.

Nakatsu, R. T. (2010). *Diagrammatic reasoning in AI*. John Wiley & Sons, Inc.

Navigli, R., & Velardi, P. (2004). Learning Domain Ontologies from Document Warehouses and Dedicated Web Sites. *Comput. Linguist.*, *30*(2), 151–179. doi:10.1162/089120104323093276

Nkambou, R. (2010). Modeling the Domain: An Introduction to the Expert Module. In R. Nkambou, J. Bourdeau, & R. Mizoguchi (Eds.), *Advances in Intelligent Tutoring Systems*. Springer Berlin Heidelberg.

Nkambou, R., Gauthier, G., & Frasson, C. (1996). CREAM-Tools: An Authoring Environment for Curriculum and Course Biulding in an Intelligent Tutoring System. In *Proceedings of the Third International Conference on Computer Aided Learning and Instruction in Science and Engineering* (pp. 186–194). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=647337.723211

Noy, N. F., & Klein, M. (2004). Ontology Evolution: Not the Same As Schema Evolution. *Knowl. Inf. Syst.*, *6*(4), 428–440. doi:10.1007/s10115-003-0137-2

Noy, N. F., & McGuinness, D. (2001). Ontology development 101: A guide to creating your first ontology. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, 1–25. doi:10.1016/j.artmed.2004.01.014

Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R. W., & Musen, M. A. (2001). Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, *16*(2), 60–71. doi:10.1109/5254.920601

Ohlsson, S. (1994). Constraint-based Student Modelling. In J. E. Greer & G. McCalla (Eds.), *Student Modelling: the Key to Individualized Knowledge-based Instruction*. Berlin: Springer.

OpenCy. (2012). OpenCyc Home page. Retrieved February 16, 2015, from http://www.cyc.com/platform/opencyc

OWL. (2012). OWL 2 Web Ontology Language Document Overview (Second Edition). Retrieved February 16, 2015, from http://www.w3.org/TR/owl2-overview/

Paquette, G. (2008). Graphical Ontology Modeling Language for Learning Environments. *Tech- Nology, Instruction., Cognition and Learning*, *5*(133–168).

Protege. (2014). The Protege Main Page. Retrieved February 16, 2015, from http://protege.stanford.edu

Py, D., Alonso, M., Auxepaules, L., & Lemeunier, T. (2008). Design of Pedagogical Feedbacks in a Learning Environment for Object-Oriented Modeling. In *Proceedings of Educators Symposium at the 11th IEEE International Conference MODELS* (pp. 39–50).

Redfield, C. L. (1996). Demonstration of the experimental advanced instructional design advisor. In *Proceedings of the 3rd International Conference on Intelligent Tutoring Systems*. Montreal, Quebec, Canada.

Roselli, R. J., Howard, L., & Brophy, S. (2006). A computer-based free body diagram assistant. *Computer Applications in Engineering Education*, *14*(4), 281–290. doi:10.1002/cae.20088

Self, J. (1988). Student models: What use are they. In P. Ercoli & R. Lewis (Eds.), *Artificial Intelligence Tools in Education*. North Holland, Amsterdam.

Self, J. (1990). Theoretical Foundations for Intelligent Tutoring Systems. *J. Artif. Intell. Educ.*, *1*(4), 3–14. Retrieved from http://dl.acm.org/citation.cfm?id=95885.95888

Semafora. (2012). Semafora Systems. Retrieved February 16, 2015, from http://www.semafora-systems.com/en/products/ontostudio/

Shapiro, S. . (1978). Path-based and node-based inference in semantic networks. In *Proceedings of the 1978 workshop on Theoretical issues in natural language processing, ACM- SIGART* (pp. 219–225).

Shute, V. J., Torreano, L. A., & Willis, R. E. (1998). DNA - Uncorking the Bottleneck in Knowledge Elicitation and Organization. In *Proceedings of the 4th International Conference on Intelligent Tutoring Systems* (pp. 146–155). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=648029.745331

Singley, M. K., Anderson, J. R., Gevins, J. S., & Hoffman, D. (1989). The algebra word problem tutor. *Artificial Intelligence and Education*, 267–275.

Sleeman, D. H., & Brown, J. S. (Eds.). (1982). *Intelligent Tutoring Systems*. Academic Press, New York.

Sottilare, R. (2012). Adaptive Tutoring & the Generalized Intelligent Framework for Tutoring, (October).

Sparks, R., Dooley, S., Meiskey, L., & Blumenthal, R. (1999). The LEAP Authoring Tool: Supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations. *International Journal of Artificial Intelligence in Education (IJAIED)*, *10*, 75–97. Retrieved from http://telearn.archives-ouvertes.fr/hal-00197337

Studer, R., Benjamins, V., & Fensel, D. (1998). Knowledge engineering: principles and methods. *Data Knowledge Engineering*, *25*(1-2), 161–197.

SUMO. (2015). The Suggested Upper Merged Ontology (SUMO). Retrieved February 16, 2015, from http://www.adampease.org/OP/

Suraweera, P., & Mitrovic, A. (2004). An Intelligent Tutoring System for Entity Relationship Modelling. *Artificial Intelligence in Education*, *14*(3-4), 375–417.

TERMINAE. (2012). Terminae Main Page. Retrieved February 16, 2015, from http://lipn.univ-paris13.fr/terminae/index.php/Main_Page

Valentine, S., Vides, F., Lucchese, G., Turner, D., Kim, H., Li, W., Hammond, T. (2012). Mechanix: A Sketch-Based Tutoring System for Statics Courses. Retrieved from http://www.aaai.org/ocs/index.php/IAAI/IAAI-12/paper/view/4857

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., … Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *Int. J. Artif. Intell. Ed.*, *15*(3), 147–204. Retrieved from http://dl.acm.org/citation.cfm?id=1434930.1434932

Weiten, M. (2009). OntoSTUDIO® as a Ontology Engineering Environment. In J. Davies, M. Grobelnik, & D. Mladenic (Eds.), *Semantic Knowledge Management: integrating ontology management, knowledge discovery, and human.* (pp. 51–60). Springer Berlin Heidelberg.

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational approaches to the communication of knowledge*. Morgan Kaufmann Pub.

CHAPTER 3

STATICSTUTOR: FREE BODY DIAGRAM TUTOR FOR PROBLEM FRAMING

This chapter is an extended version from a paper published in *the 12th International Conference on Intelligent Tutoring Systems, with author list:*

Enruo Guo, Stephen Gilbert, John Jackman, Gloria Starns, Mathew Hagge, LeAnn Faidley, Mostafa Amin-Naseri

Enruo Guo's roles in this research included substantial contributions to 1) the conception and design of the tutor, namely the development of the tutoring system underlying the diagramming tool (the domain-wide check functions, evaluation process, and feedback generation), and 2) collection of diagnostic student data from the system. She wrote the following portions of this chapter: 3.2 Related Work, 3.3 StaticsTutor Interface and Architecture, half of 3.4 Preliminary Evaluation, and 3.5 Conclusions and Future Work.

Abstract

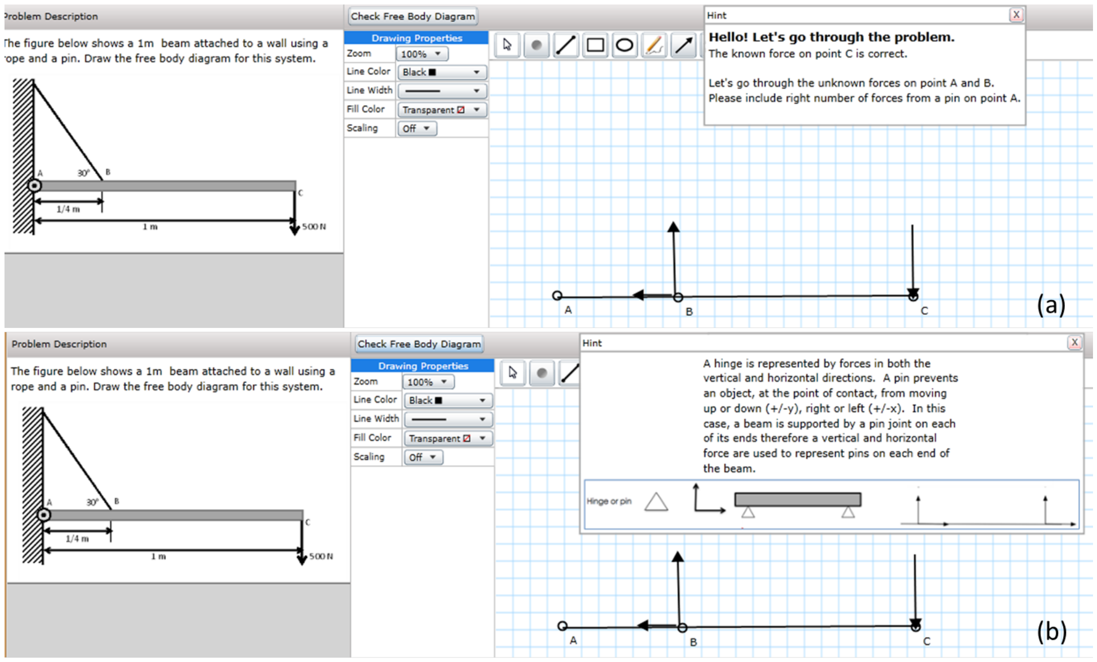
While Intelligent Tutoring Systems have been designed to teach free-body diagrams, existing software often forces students to define variables and equations that may not be necessary for conceptual understanding during the problem framing stage. StaticsTutor was developed to analyze solutions from a student-drawn diagram and recognize misconceptions at the earliest stages of problem framing, without requiring numerical force values or the need to provide equilibrium equations. Preliminary results with 81 undergraduates showed that it detects several frequent misconceptions in statics and that students are interested in using it, though they have suggestions for improvement. Data suggested that students who rate the tutor feedback most helpful are those who need it the most. This research offers insights in the development of a diagram-based tutor to help problem framing, which can be generalized to tutors for other forms of diagrams.

Keywords: intelligent tutoring system, statics tutor, free-body diagram.

3.1.Introduction

If you are one of the over 100,000 freshman engineering students in the U.S. (National Science Foundation & National Center for Science and Engineering Statistics, 2013), you will likely take a course in engineering mechanics called statics. Streveler, et al.'s (2008) elegant overview of conceptual learning within engineering notes that statics, along with thermal science and electrical circuits, is one of the most difficult learning domains. Chi (2005) and Reiner, et al. (2000) address the question of why some misconceptions are particularly prevalent and difficult to correct. Their results suggest that particularly problematic misconceptions may be based on metaphors to physical phenomena that are similar but not quite right, e.g., thinking of electrical current as a fluid. Or, difficult misconceptions are based on phenomena with unobservable components or relationships. Computer simulations are mentioned as a potential solution. This paper describes the StaticsTutor, an attempt to provide students not only with the simulation, but also with interactive feedback that directly addresses conceptual challenges. StaticsTutor was developed as part of a problem framing research project funded by the National Science Foundation (EEC-1025133). This paper describes an initial assessment of the StaticsTutor's usability and an exploration of the dynamics of its usage with 81 engineering undergraduates.

A student assigned a homework problem using StaticsTutor sees a problem statement and a descriptive figure for the statics problem shown on the left side of a web-based interface (Figure 3.1). The student begins by first drawing the overall problem information in a drawing area to the right of the problem statement, i.e., a beam, its support forces, external forces, and/or moments. The drawing software offers a typical drawing palette but has been customized for engineering. It includes tools to create force vectors, moments, labels, coordinate systems, etc. A point object is placed on the beam to show the location of supports and externally applied forces. The student must complete the free body diagram required by the problem, using the given forces to solve for the unknown forces.



Problem Description

The figure below shows a 1m beam attached to a wall using a rope and a pin. Draw the free body diagram for this system.

Zoom: 100%
Line Color: Black ■
Fill Color: Transparent
Scaling: Off

Hint

Hello! Let's go through the problem.
The known force on point C is correct.

Let's go through the unknown forces on point A and B.
Please include right number of forces from a pin on point A.

Hint

A hinge is represented by forces in both the vertical and horizontal directions. A pin prevents an object, at the point of contact, from moving up or down (+/-y), right or left (+/-x). In this case, a beam is supported by a pin joint on each of its ends therefore a vertical and horizontal force are used to represent pins on each end of the beam.

Figure 3.1. A rope and pin problem. (a) A pop-up window summarizes problem-specific feedback. (b) A pop-up window is triggered by a potential misconception and gives an example on how a pin/hinge reacts in a system and how the reaction forces should be represented.

At any point while working on the problem, the student may click a button labeled, "Check Free Body Diagram," which invokes the tutor. The tutor provides feedback in a popup window based on comparing the student's diagram with a known solution to this problem and with a set of general rules about free body diagrams that apply across problems. The student receives feedback at two conceptual levels. First, are all the pieces of the diagram present? Second, are there any likely misconceptions?

The student's tutoring experience described above raises the following research questions. 1) How is StaticsTutor different than existing tutors for science and engineering diagrams? 2) How do students engage with StaticsTutor, and do they find it usable and useful? 3) Does StaticsTutor help prevent misconceptions on future problems in the real world? In this initial research the first two questions are addressed.

3.2.Related Work

It is well established that engineering problem solving skills are critical for students to become practicing engineers. The most important stage of problem solving is problem framing, which occurs at the onset of problem solving in which students structure the problem using reasoning and metacognitive skills (Diefes-Dux, H. A., Salim, 2009; Redish, E. F., Smith, 2008). When they know to do so, students typically attempt framing as the first step of the problem solving process (Liikkanen & Perttula, 2009; Litzinger, Lattuca, Hadgraft, & Newstetter, 2011), and Voss & Post (1988) found that early framing leads to better success in later stages of the problem.

A model-tracing tutor can be successful for well-defined problem-solving procedures, but recognizing the student’s model for solving an open-ended engineering challenge is a work in progress. In the last a few decades, Intelligent Tutoring Systems (ITSs) have steadily improved to make content more accessible to the average students (Koedinger et al., 1997; Sottilare, Goldberg, Brawner, & Holden, 2012). Tutors have been used both in class and for homework in mathematics, physics, computer programming, and other subjects (Anderson et al., 1995; Corbett, Koedinger, & Hadley, 2002; Johnson & Holder, 2010; Vanlehn et al., 2005). While these systems have been successful, many do not explicitly tutor on underlying concepts, focusing instead on helping students master the procedural skills needed to solve problems. StaticsTutor was designed with the intent to distract students as little as possible with numerical values and focus on the concepts of statics: equilibrium, resolution of forces into their orthogonal components, and summation of moments about any moment axis consistently using the rules of vector multiplication.

A variety of previous ITSs, interactive media, and YouTube videos have addressed free body diagrams. A YouTube search on "free-body diagram physics" reveals a few hundred results, most of which are simple tutorials (Gmuchomas, 2009; PhysicsEH, 2008). Interactive media and simulations are available as well: MasteringPhysics (Pearson, 2015) is a website accompanying Pearson physics textbooks that helps students solve their physics homework. There are also some Java applets that teach free body diagram drawing, and calculate the net force on an object, e.g., Hwang (2004). Force Effect by Autodesk enables students to create models on an iPad and subsequently acquire equations of equilibrium. However, none of these multimedia tools are ITSs that are able to communicate interactively with students and help them learn to solve a series of problems.

In the domain of intelligent tutors, we can expect more personalized feedback and conceptual teaching. Evaluations have shown that Andes (Vanlehn et al., 2005) helps students learned more significantly comparing doing homework on paper. Unlike other systems, Andes requires students to enter intermediate steps, such as defining variables, drawing vectors, etc. Feedback is provided after each step. Hints are also available during the middle of problem solving process.

There are some other existing diagram-based tutoring systems. COLLECT-UML (Baghaei & Mitrovic, 2005) supports students learning UML class diagram either individually or collaboratively. EER-Tutor (Zakharov et al., 2005) helps learning and practicing principles of Enhanced Entity-relationship modeling. Free-Body-Diagram Assistant (Roselli & Howard, 2003) provides students opportunities to construct FBDs for the human body and receive constructive feedback in biomechanics. CogSketch (Forbus et al., 2011), which is a sketch-based educational software application, has demonstrated the powerful ability to understand sketched shapes and recognize them even after rotation or change of position. Labeling provides a rapid way to match instructor and student components. However, in engineering statics, many problems ask the student to define one or more forces without requiring specific labels on those forces, so matching by labels has some limitations. Mechanix (Valentine et al., 2012) is a free-body diagram tutoring system based on free-hand drawing recognition. A checklist area is shown with specific instructions to guide students in order to finish the problem, such as drawing the object from the picture, drawing the axis, given forces and reaction forces. However, a typing mode is still needed to check the value of each force. Unlike the existing free body diagram tutors, StaticsTutor addresses conceptual understanding at the problem framing stage.

3.3.StaticsTutor Interface and Architecture

The tutor uses a web-based drawing interface, XDraw, developed internally by author John Jackman using the Microsoft Silverlight framework. A backend database saves students diagrams. StaticsTutor communicates with XDraw via a TCP socket between the two servers. Currently, authoring is based on xml files on the tutor server. In the future, a GUI will be implemented to serve as an authoring tool and interface. XDraw supports basic drawing objects such as points, lines, rectangles and vectors as well as free-hand drawing. A coordinate system is an object that is defined by the drawing tool as well. Students can rotate the coordinate system to facilitate solving the problem so that the angle of the forces would be adjusted based on the new axes.

The drawing can be designated as scaled or un-scaled. A scaled drawing allows the students to set up a grid scale for distance and the magnitude and units of force vectors. Also, a measuring tool is provided to measure distance between any two points. The scaled mode can be applied to vector magnitude check and distance measurement. The un-scaled mode provides more flexibility in creating a free-body diagram at the problem framing stage. For this study, students used the un-scaled mode to solve problems.

The StaticsTutor architecture is shown in Figure 3.2. Components labeled "Problem…" contain information specific to the particular problem, while the domain-wide components are used across statics problems. This overall architecture has been used by a tutor in thermodynamics courses as well, the Thermo Cycle Tutor (Guo et al., in preparation), which indicates the generality and feasibility to different domains.

The tutor includes five parts. The problem solution, created by an expert instructor, contains a correct diagram and appropriate force values. The tutor can designate the student correct if the student's diagram is conceptually equivalent to the problem solution even if not identical. E.g., for a pin, the force could be pulling from below or pushing from above and both would be correct. Similarly, point B in Figure 3.1 must be between

A and C, and not too close to either side, but its exact position is not important. The problem solution contains the number of forces reacting at each point, and the angle of each force. Magnitudes of forces are not represented because this approach is focused on conceptual structure. The tutor can accept all the possible correct angles. A tolerance value for the angles can be added by the solution author to make the tutor more accepting of student vectors that are not exact, e.g., 10 degrees. Also, the tutor gives students the freedom to draw either resultant forces or resolved forces in a selected orthogonal coordinate system.

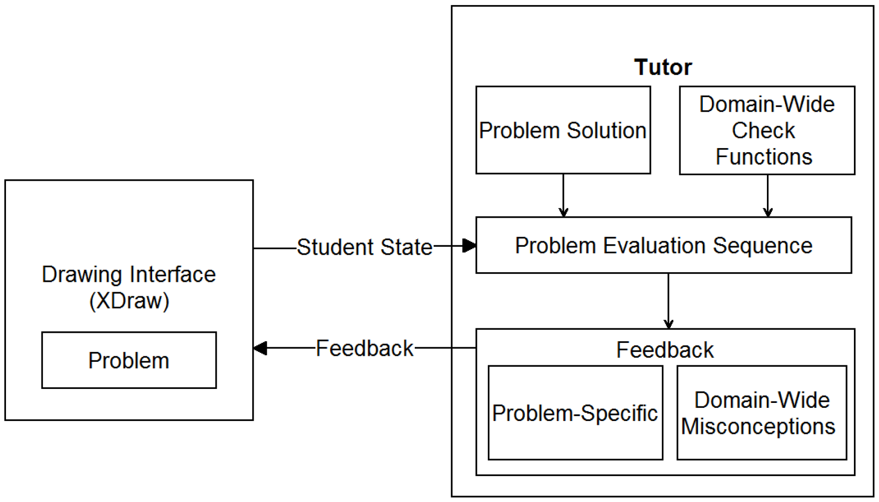

Tutor
Problem Solution
Domain-Wide Check Functions
Drawing Interface (XDraw)
Problem
Student State
Problem Evaluation Sequence
Feedback
Feedback
Problem-Specific
Domain-Wide Misconceptions

Figure 3.2. The architecture of StaticsTutor. The boxes labeled "Problem…" are specific to the problem at hand, while the "Domain-Wide" components are used across statics problems.

StaticsTutor's problem evaluation sequence evaluates the free-body diagram in a number of steps inspired by the three overall stages of problem framing defined by several authors (Cuban, 1990; K. A. Leithwood & Stager, 1989; K. Leithwood & Steinbach, 1992). First, the learner defines the stated problem. Second, the learner reflects on the stated problem, which involves a) a review of his or her personal assumptions about the problem situation, b) identification of a clear interpretation of the problem, and c) identification of preexisting solutions embedded in the problem situation defined at stage a. Third, the learner reframes the problem if necessary.

In the example problem in this paper (Figure 3.1), the StaticsTutor problem evaluation sequence contains the following eight steps. A beam (1 meter) is attached to a wall using a pin (point A) and a rope (point B). An external force (500 Newtons) acting at point C is pushing downward on the beam. The weight of the beam can be neglected since the problem statement does not indicate otherwise. Initially, the tutor takes three steps to consider if the student has clear recognition of the stated problem, which contains check of whether all non-force components are present, and a check of the given forces: Step_1 “if a beam is present and point A, B and C are present and in correct relative location,” Step_2 “if the number of forces at point C is correct,” and Step_3 “if the angle of force at point C is correct.” These three steps correspond to the first stage of problem framing: defining the problem. Each step functions by calling one or more of the domain-wide check functions. E.g., a check function that accepts a diagram point and the number of expected forces at that point per the problem solution might look like *numForcesCorrect(point, expectedNumForces)* and return a *True* or *False*. More detail on check functions provided below.

The tutor's next steps (4-7) focus on second stage in problem framing: whether the student has a clear interpretation of the problem. It contains Step_4 “if the number of forces at point A is correct,” Step_5 “if the angle of the forces at point A are correct,” Step_6 “if the number of forces at point B is correct,” and Step_7 “if the angle of the forces at point B are correct.” If any of the angles of the student diagram do not match the problem solution, the tutor will offer problem-specific feedback, such as, "Please check the angle of the force at point A." Also, the tutor evaluates the student's interpretation of the problem by evaluating the diagram components with a pool of domain-wide misconceptions. If the pin (point A), for example, does not have both its vertical and horizontal components represented, then the student may not understand that a pin exerts forces in both directions, and would receive the misconception feedback for pins. There are similar misconception feedback checks for ropes, hinges and rollers. Domain-wide misconception feedback contains texts and pictorial explanations created by co-authors Starns and Faidley, who teach engineering statics. The feedback has been used in multiple problems.

The last stage is to check student's reframing of the problem. To check for reframing, StaticsTutor looks for extra information that may have drawn initially before the student recognized the type of problem appropriately. Step_8 checks for anomalies such as a force that is not associated with any point or line, or an extra point that is not needed. Each of the eight steps needs to be satisfied for the problem to be complete.

Note that the eight steps of the problem evaluation sequence described above are specific to the particular problem posed, although the evaluation functions they used are the domain-wide check functions. Therefore, problem authors can change the evaluation sequence based on the needs of the problem or based on different pedagogical preferences, though most likely the sequence will still correspond to the three stages of problem framing. For example, the check for extra information might be removed if the instructor is not interested in this sort of evaluation. Also, some experts do not require students to draw the beam if the beam's weight is negligible, in which case they might chose to remove the beam check.

It is important to describe the check functions in more detail because they support StaticsTutor's approach that is agnostic of force values, enabling the more generic problem framing analysis. A pool of domain-wide check functions is available to the problem evaluation sequence. This pool contains general checks applicable to all problems, such as check the angle of a force, check the number of forces attached to a point, etc. Check functions were designed in three levels to handle forces. First, a force can be directly accessed via its label, if it has been pre-defined by the problem statement, i.e., F_1, and the student has labeled it. Second, a force can be indirectly accessed via its contacting point, e.g., a pin. In most situations, the name of the contacting point is given in the problem statement. Students have the flexibility to draw forces attached to contacting points and name the forces to their liking. By this means, StaticsTutor does not need to enforce labeling of the forces in order to ensure a match with the expert solution. Third, a force can be attached to other objects, e.g., a line, where it might represent the weight of a beam. StaticsTutor gives point association a higher priority than other object association. So a force is considered as *object-associated* only if it is not attached to any point.

Lastly, a student model is constructed for the purposes of tracking student performance, recording the student’s misunderstandings and facilitating the instructor’s analysis. The student model contains: 1) the series of student drawings, each of which is automatically saved when the student sends a request for feedback; 2) the feedback message generated by the tutor; and 3) answers to a post-survey that is administered using third-party software.

3.4. Preliminary Evaluation

StaticsTutor was tested on 81 engineering undergraduates in Fall 2013 who were enrolled in the first-year mechanical engineering courses. Pre-requisites for the two courses were trigonometry proficiency and concurrent enrollment in the first semester calculus course. A statics problem was offered as a homework assignment using StaticsTutor. Students were allowed to do the homework anytime in a one-week period on their own PC or laptop. In addition to the drawing activity, an initial pilot test of an equilibrium table activity was being conducted along with the main activity. After students finished their drawing, they were guided to an equilibrium table with its own tutor that helped students to write their equations and find the unknown values. However, the focus of this study is only the diagram part of the tutor. After the students finished the problem, they then completed a survey with some background questions as well as some user experience questions. The user experience survey was adapted from Lewis' (1995) CSUQ questionnaire and used a five-point Likert scale, with 1 being strongly disagree and 5 strongly agree.

Each time a student requested feedback from the tutor, his/her drawing was saved and was evaluated per the 8-step evaluation sequence described above. In total, all students requested feedback 714 times. Different methods of failure during these sequences were categorized in Table 3.1. Looking at students' mistakes, 25.5% of the feedback focused on missing the basics (Step_1). Most students mastered those basics and continued. Few students need help placing the forces at point C. However, about 50% of the total feedback was related to the hinge and rope points, known potential misconceptions. Again, most students continued after feedback. The small percentage of *Fully_Correct* evaluations is to be expected, since there would be one per student.

It is worth noting the dynamics of these feedback requests. About 24% of students solved the problem completely before their first request for feedback, so that the request simply confirmed that they were finished. The remaining students made between 2 and 32 requests, with a mean of 6.8 and median of 4. Their time to complete the problems waw similarly distributed with a maximum time of 44 minutes, a mean of 7.8 minutes, and median of 3.8 minutes. Finally, out of the 81 participants, only 64 students successfully completing the problem (79%). The non-completing students' mean time on the problem was 5.5 minutes.

Table 3.1. Methods of Failure: Categories of Tutor Feedback Across 714 Requests.

Basics missing	Forces at C missing	Hinge & Rope	Rope Issue	Hinge Issue	Extraneous Info	Fully Correct
25.5%	2.8%	15.7%	16.1%	18.9%	1.4%	19.6%

Comparing the students' background information with their performance reveals that we could likely benefit from customizing the tutoring experience for different skill levels. Students who had taken physics or statics before had lower problem solving times and fewer feedback requests on average. Also, among the students who completed the problem on their first submission (i.e., without any feedback) all of them had at least one of the two courses.

Students were asked to complete survey questions after using XDraw. The survey was consisted of four main categories of questions: nine questions on usability, three questions about their productivity using the tutor, two on whether they believe they had learned things in the activity, and three other question about the quality of the feedback they were given. The mean usability score was 3.5 (SD 1.11), which shows an overall positive experience with the tutor. The mean for each usability question score is shown in Figure 3.3. Students were slightly positive, mean 3.04 (SD 1.08) on whether they believed they learned new things in the activity. However, they didn't feel as positive about their productivity using the tutor: mean 2.86 (SD 1.17). A likely explanation for this is that the accompanying pilot test of the equilibrium table was not well synchronized with the drawing, and presented some students with frustrating technical issues. This issue may also have had a negative impact on the students' estimation of feedback quality, mean 2.73 (SD 1.08). However, in free response comments, several students mentioned a desire for more specific feedback that would help them better understand what was wrong with their drawing. The misconception feedback was intended for that purpose, but can still be improved.

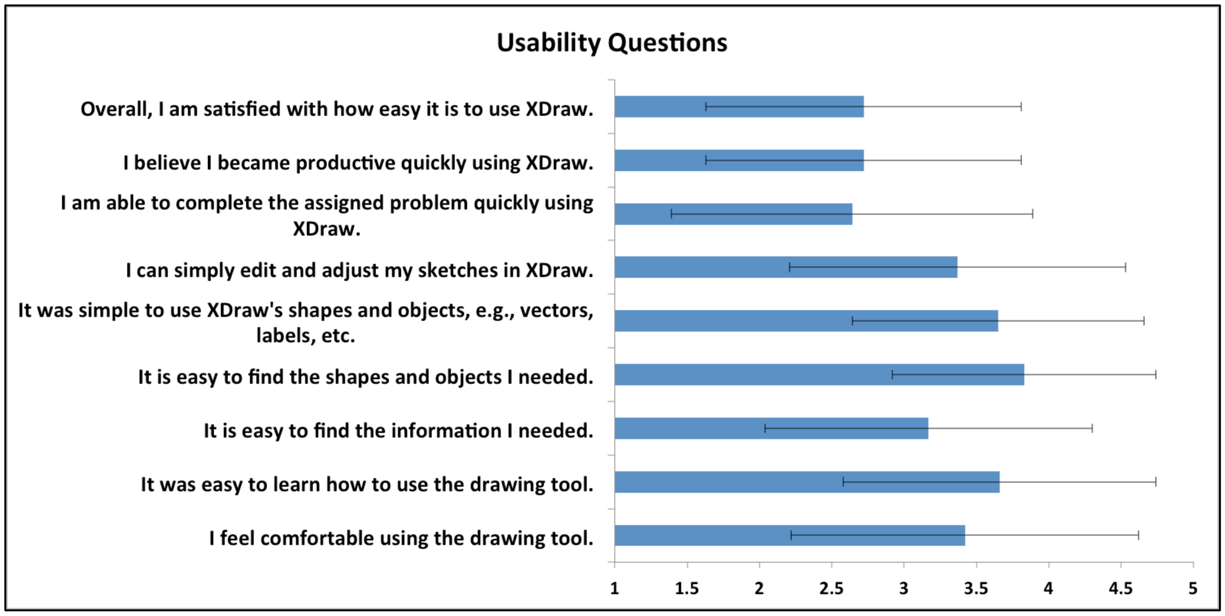


Figure 3.3. Means on usability scores.

There was an interesting pattern in rating the quality of feedback. Three Likert questions were asked, e.g., "The feedback is easy to understand." Unexpectedly those students who were unable to finish the problem, on average, rated the feedback better on all three questions than those who completed the activity. These differences appear statistically significant, but the data fail the normality assumption, and the Mann-Whitney U test did not reveal significance. However, it was a notable trend that may arise because the tutor was designed for students who have difficulty in problem solving, and therefore that group found it more helpful. More data are needed to confirm.

3.5. Conclusions and Future Work

As most existing ITSs require students to define variables and equilibrium equations that may not be necessary during the problem framing stage, there is a potential contribution by StaticsTutor. Its evaluation is based on the three subskills in problem framing: 1) defining, 2) reflecting and 3) reframing the problem. It can evaluate a free-body diagram without requiring labeling of the forces, and can detect misconceptions based on each problem component. The architecture of StaticsTutor provides the instructor the freedom to choose the procedure and pieces he or she would like to evaluate and also gives students the option of either drawing resultant forces or decomposing them using a specified coordinate system.

Future work will integrate the ability to customize feedback based on the student model data carried from problem to problem. Results from student surveys also indicated that students are not yet satisfied with the overall productivity of StaticsTutor. Improvements to the StaticsTutor interface to facilitate the problem framing and drawing stages continue to be incorporated and are based on feedback from both expert users and students.

REFERENCES

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences*, *4*(2), 167–207.

Baghaei, N., & Mitrovic, A. (2005). COLLECT-UML: Supporting Individual and Collaborative Learning of UML Class Diagrams in a Constraint-based Intelligent Tutoring System. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part IV* (pp. 458–464). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/11554028_64

Chi, M. T. H. (2005). Commonsense Conceptions of Emergent Processes: Why Some Misconceptions Are Robust. *Journal of the Learning Sciences*, *14*, 161–199.

Corbett, A. T., Koedinger, K. R., & Hadley, W. S. (2002). Cognitive Tutors: From the research classroom to all classrooms. In P. S. Goodman (Ed.), *Technology enhanced learning: Opportunities for change* (pp. 235–263). Mahway, NJ: Lawrence Erlbaum Associates.

Cuban, L. (1990). *Problem-finding: Problem-based learning project*.

Diefes-Dux, H. A., Salim, A. (2009). Problem Formulation during Model-Eliciting Activities: Characterization of First-Year Students' Responses. In *Proceedings of the Research in Engineering Education Symposium 2009*. Palm Cove, Queensland, Australia.

Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2011). CogSketch: Sketch Understanding for Cognitive Science Research and for Education. *Topics in Cognitive Science*, *3*(4), 648–666. doi:10.1111/j.1756-8765.2011.01149.x

Gmuchomas. (2009). Free body diagrams. Retrieved April 4, 2015, from http://www.youtube.com/watch?v=TFgKj4w35BM

Guo, E., Jackman, J., Gilbert, S., Hagge, M., Starns, G., Faidley, L., & Amin-Naseri, M. (in preparation). *Decision-centric problem framing tutor for thermodynamics*.

Hwang, F.-K. (2004). Free body force diagram. Retrieved April 4, 2014, from http://www.thephysicsfront.org/items/detail.cfm?ID=1870

Johnson, B. G., & Holder, D. A. (2010). A Model-Tracing Intelligent Tutoring System for Assigning Oxidation Numbers in Chemical Formulas. *Chemical Educator*, *15*, 447–454.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent Tutoring Goes To School in the Big City, 30–43.

Leithwood, K. A., & Stager, M. (1989). Expertise in Principals' Problem Solving. *Educational Administration Quarterly*, *25*, 126–161.

Leithwood, K., & Steinbach, R. (1992). Improving the Problem-Solving Expertise of School Administrators: Theory and Practice. *Education and Urban Society*, *24*(317-345).

Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human ‐ Computer Interaction*, *7*, 57–78.

Liikkanen, L. A., & Perttula, M. (2009). Exploring problem decomposition in conceptual design among novice designers. *Design Studies*, *30*(1), 38–59. doi:http://dx.doi.org/10.1016/j.destud.2008.07.003

Litzinger, T., Lattuca, L. R., Hadgraft, R., & Newstetter, W. (2011). Engineering Education and the Development of Expertise. *Journal of Engineering Education*, *100*(1), 123–150. doi:10.1002/j.2168-9830.2011.tb00006.x

National Science Foundation, & National Center for Science and Engineering Statistics. (2013). *Women, Minorities, and Persons with Disabilities in Science and Engineering*. Arlington, VA. Retrieved from http://www.nsf.gov/statistics/wmpd/

Pearson. (2015). Mastering Physics. Retrieved April 4, 2015, from http://www.masteringphysics.com

PhysicsEH. (2008). Free body diagrams physics help. Retrieved April 4, 2015, from http://www.youtube.com/watch?v=BuPfDI7TyL0

Redish, E. F., Smith, K. A. (2008). Looking beyond content: Skill development for engineers. *Journal of Engineering Education*, *97*(3), 295–307.

Reiner, M., Slotta, J. D., Chi, M. T. H., & Resnick, L. B. (2000). Naive Physics Reasoning: A Commitment to Substance-Based Conceptions. *Cognition and Instruction*, *18*, 1–34.

Roselli, R. J., & Howard, L. (2003). Integration of an Interactive Free Body Diagram Assistant with a Courseware Authoring Package and an Experimental Learning Management System. In *Proceedings of the American Society for Engineering Education Annual Conference*. Nashville, TN.

Sottilare, R. a, Goldberg, B. S., Brawner, K. W., & Holden, H. K. (2012). A Modular Framework to Support the Authoring and Assessment of Adaptive Computer-Based Tutoring Systems (CBTS), (12017), 1–13.

Streveler, R., Litzinger, T., Miller, R. L., & Steif, P. S. (2008). Learning conceptual knowledge in the engineering sciences: Overview and future research directions. *Journal of Engineering Education*, *97*(3), 279–294. doi:10.1002/j.2168-9830.2008.tb00979.x/

Valentine, S., Vides, F., Lucchese, G., Turner, D., Kim, H., Li, W., … Hammond, T. (2012). Mechanix: A Sketch-Based Tutoring System for Statics Courses. Retrieved from http://www.aaai.org/ocs/index.php/IAAI/IAAI-12/paper/view/4857

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., … Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *Int. J. Artif. Intell. Ed.*, *15*(3), 147–204. Retrieved from http://dl.acm.org/citation.cfm?id=1434930.1434932

Voss, J. F., & Post, T. A. (1988). *On the solving of ill-structured problems.* Hillsdale, NJ, England: Lawrence Erlbaum Associates, Inc.

Zakharov, K., Mitrovic, A., & Ohlsson, S. (2005). Feedback Micro-engineering in EER-Tutor. In *Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning Through Intelligent and Socially Informed Technology* (pp. 718–725). Amsterdam, The Netherlands, The Netherlands: IOS Press. Retrieved from http://dl.acm.org/citation.cfm?id=1562524.1562620

CHAPTER 4

A DECISION-CENTRIC PROBLEM FRAMING TUTOR FOR THERMODYNAMICS

This chapter will be submitted to the journal *Computers and Education* with author list:

Enruo Guo, John Jackman, Stephen Gilbert, Mathew Hagge, Gloria Starns, LeAnn Faidley, Mostafa Amin-Naseri

Enruo Guo's role included: 1) develop the domain model, expert module and evaluation engine of the tutoring system; 2) collect diagnostic data; 3) write most parts of the paper, except the Introduction and 4.2.2 and 4.2.3 within Related Work.

Abstract

We present Thermo Cycle Tutor, a Decision-centric tutoring system to help engineering undergraduates during problem framing stage in a thermodynamics course. By implementing a set of core decisions in a refrigeration cycle, and pedagogical instructions from a thermodynamics expert, the tutor has shown its effectiveness in improving students' test scores. The tutor contains eight components: drawing interface, domain model, expert module, evaluation engine, pedagogical module, student model, UI controller, and training materials. Features about the tutor include: auto-generation of expert solution based on domain rules and problem-specific input; interpretation of student's drawing in a sophisticated way so that procedural knowledge and conceptual knowledge can be handled differently; a pool of check functions that allows knowledge reuse and re-organization for future authoring.

4.1. Introduction

Chen et al, (2010) reported on the positive effects of rapid feedback on student learning in a classroom scenario for a statics course. Students answered multiple choice quiz questions using hand-held devices followed by immediate feedback on their answers. Students could also see the aggregate data on all the student answers. The results were used in real time by the instructor to identify misconceptions and poor understanding of the material in order to provide a more focused explanation of the concepts and principles. Similar approaches have been implemented using clickers to provide immediate feedback in classrooms (Bursic, 2012; Patterson, B., Kilpatrick, J., Woebkenberg, 2010; Perkins, K. K., Turpen, 2009). Both Bursic and Patterson et al. found that feedback increased the level of student engagement in class but had no significant effect on student learning.

Shute (2008) has reviewed a variety of formative feedback approaches that have been used to improve student learning, finding that the effects of feedback on student learning are inconsistent. It was recommended that feedback should be targeted for the individual and not just the task at hand. Formative assessment was described as an evaluation performed on an individual basis that was designed to change student thinking and behavior. Hattie and Timperley (2007) analyzed multiple feedback approaches and claimed that, effective feedback needs to be clear, purposeful and meaningful. It should be compatible with students' prior knowledge with some logical connections.

They also recommended against providing feedback for students who have a poor understanding of the underlying concepts but rather a focused instructional effort to help student learning. Traditionally, feedback is given informally (e.g., answering questions in class or during office hours) and formally (e.g., assignments or exams). The difficulty in this approach is scalability and frequency, as the number of students in a class increases. When multiple instructors are involved there is often significant variability in the quality of feedback.

Researchers from the Pittsburgh Advanced Cognitive Tutor Center and the Science of Learning Center at Carnegie Mellon have pioneered the development of cognitive tutors that provide feedback with an emphasis on “learning-by-doing.” This work has led to tutor software in mathematics for secondary education that has been widely used and is based on Adaptive Control of Thought-Rational (ACT-R) theory of human cognition(Anderson, 1996). A review of the research literature on the effectiveness of this approach reported inconsistent effects on student learning (Education, 2013).

The Open Learning Initiative (OLI) at Carnegie Mellon University provides a set of courses that consist of multimedia content partitioned into modules (Lovett, M., Meyer, O., Thille, 2008; Steif, P. S., A. Dollár, 2009). Students receive immediate feedback from questions (e.g., multiple choice, short answer) that are presented at regular intervals throughout a module. In addition, end-of-module exercises provide a summative assessment of what the student has learned in the module. Students proceed in a linear fashion through the modules. Their research has shown that students using OLI instruction achieved the same or better learning outcomes in much less time than students taking a traditional course.

One of the National Academy of Engineering grand challenges is to advance personalized learning (Engineering, 2008). It is widely recognized that current practices of delivering standard lectures and assessments do not address the needs of a diverse body of students that come to a class with very different levels of understanding and preparation. To provide a personalized learning environment, a model of student understanding is needed so that instruction can be customized for each student who is on a different learning trajectory.

The most important stage of problem solving is problem framing, which occurs at the onset of problem solving in which students structure the problem using reasoning and metacognitive skills (Diefes-Dux, H. A., Salim, 2009; Redish, E. F., Smith, 2008). We have found that students' ability to frame a problem is inextricably linked to their grasp of underlying concepts and principles associated with problem elements. Therefore, developing a more general theory of problem framing that can lead to significant improvements in engineering problem solving skills must consider the relationship of these skills to students' understanding of the relevant concepts and principles.

Instead of solving large numbers of problems to gain expertise, students that develop good problem framing skills should be able to achieve expertise with smaller problem sets. Without good problem framing skills, students will resort to poor strategies such as formula matching or memorizing similar problems, which will be insufficient when dealing with new problems that they have not seen. Our premise is that students will require less problem solving practice with more explicit decision making during problem framing and become domain experts much faster, because their decision making, understanding, and thought processes are much more like that of an expert when they have internalized a more explicit decision making process.

4.2. Related Work

4.2.1. Intelligent Tutoring Systems (ITSs)

ITSs are computer-based systems, which contains learning contents as well as pedagogy strategies to teach (Wenger, 1987). It is designed to simulate human teacher's behavior and guidance after interpreting student's responses. Currently, ITSs have shown their educational effectiveness in many areas. Its one-on-one tutoring paradigm gives students more personalized guidance. ITSs have been used both in class and for homework in mathematics, physics, computer programming, and other subjects (A. T. Corbett et al., 2002; Oliveira Neto & Nascimento, 2012; Rai & Beck, 2012; Sklavakis & Refanidis, 2013; Vanlehn et al., 2005). Also, ITSs was incorporated in e-learning systems to provide personalized courses of action based on student's profile (Dolenc & Aberšek, 2015; Schiaffino, Garcia, & Amandi, 2008). To benefit ITSs' one-on-one tutoring paradigm, many systems have been developed to teach students how to draw a diagram. A few examples are listed. COLLECT-UML (Baghaei & Mitrovic, 2005) supports computer science students to learn UML class diagrams either individually or collaboratively. EER- Tutor (Zakharov et al., 2005) helps learning and practicing principles of Enhanced Entity-relationship modeling in database course. CogSketch (Forbus et al., 2011), which is a sketch-based educational software, has demonstrated the powerful ability to understand sketched shapes and recognize them even after rotation or change of position. Free-Body-Diagram Assistant (Roselli & Howard, 2003) allows students to construct FBDs on a given human body picture and receive constructive feedback in the course of biomechanics. StaticsTutor (Guo et al., 2014) is a Free-body diagram tutoring system for engineering undergraduates with emphasis on student's conceptual understanding at problem framing stage.

In general, a Diagram-based ITS includes 1) User interface that usually contains a drawing palette; 2) Pedagogical module which incorporates expert's feedback and pedagogical strategies; 3) Domain model that contains semantic or syntax constraints or knowledge applicable to all problems in the system; and 4) Student model. Given the varieties of different ITSs structures, some of them have a diagnosis module, which compares student's diagram with expert's solution, if the domain model is unable to automatically generate a solution. If student's drawing doesn't agree with the ideal drawing, an instructional feedback or hint will be given. Because the systems are designed to interpret varied student' responses, they are able to determine why student's understanding has gone astray. Thus, personalized feedback is offered to fix their misconception or misunderstanding at the earliest time.

4.2.2. Decision-centered pedagogy

The process of formulating a general to specific decision set, learning about individual student understanding, and helping students make decisions through their current understanding is called decision based learning (DBL). The goal of DBL is to allow students to solve problems without full expertise (like a novice), and to improve conceptual understanding and expertise through the decision making/instructor feedback loop. Students' decisions give the instructor specific knowledge about the students, and allow the instructor to provide individualized feedback in the form of additional questions or activities. The instructor feedback is designed to develop students' understanding to the point where they can make the correct decision, based solely on the students' thought process (Figure 4.1). This process is ideally suited to a tutoring system because the decision making process reveals information about each student, the success or failure of the instructor's help, and the effectiveness of the instructor's decision set in creating understanding that can be applied to all future problems.

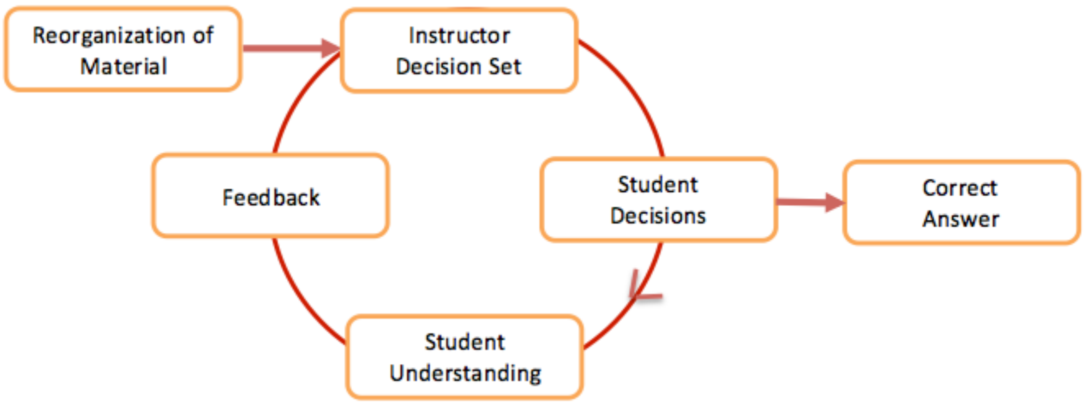


Figure 4.1. The Decision-based Learning (DBL) process. From Hagge et al. (submitted).

4.2.3. Thermodynamics domain

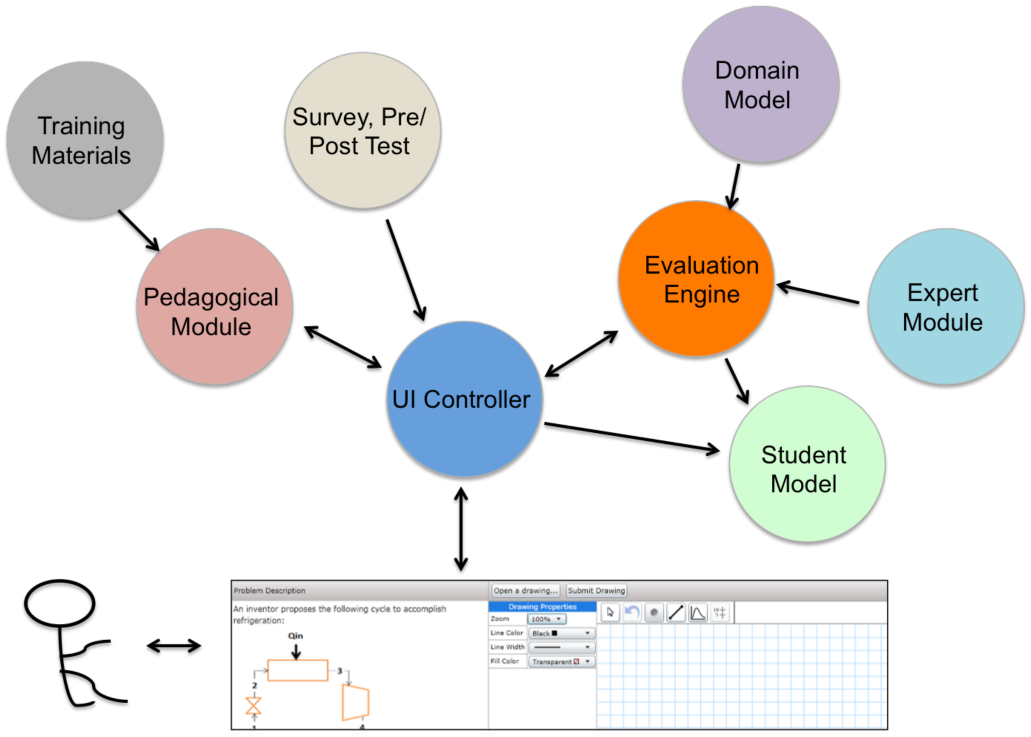
It is well established that thermodynamics is a major challenge for undergraduate engineering students in terms of their ability to understand and properly use the concepts and principles at a level that is necessary to solve engineering problems (Miller et al., 2006; Prevost, Haudek, Urban-Lurain, & Merrill, 2012; R. A. Streveler, Litzinger, Miller, & Steif, 2008). Chi (2005) and Reiner, et al. (2000) address the question of why some misconceptions are particularly prevalent and difficult to correct. Their results suggest that particularly problematic misconceptions may be based on metaphors to physical phenomena that are similar but not quite right, e.g. thinking of electrical current as a fluid. Computer simulations are mentioned as a potential solution. One approach to improve student understanding of thermodynamics was to create more interactive instructional materials using computer-based instruction to supplement traditional textbooks (Anderson & Taraban, 2005; Taraban, Anderson, Sharma, & Weigold, 2003). Interactions included answering multiple choice and short answer questions as well as controlling simulations of devices and system behaviors. To identify students' misconceptions about thermodynamics, Beall (1994) posed conceptual questions to the students and asked them to write a response. This assessment of student understanding was used to clear up misconceptions in subsequent lectures. Building on this work, Prevost et al. (2012) scaled the approach to large classes using automated text analysis to provide instructors with an analysis of students' constructed responses. However, none of their approaches rely on computer-based intelligent systems to detect misconceptions through diagrams in thermodynamics problems.

The remainder of this paper is organized as follows: In Section 4.3, we describe the tutor architecture. Section 4.4 illustrates the tutor operation using an example session, and conclusions are drawn in Section 4.5.

4.3. Tutor Architecture

Thermo Cycle Tutor uses a web-based drawing interface, XDraw, developed by author Jackman using the Microsoft Silverlight framework. Communication between the drawing tool and tutor server is based on TCP Socket. A backend database saves students diagrams and activity logs. XDraw supports basic drawing objects such as points, lines, rectangles and vectors as well as free-hand drawing. It was used as the user interface for StaticsTutor (Guo et al., 2014) in statics course before. By adding some domain-specific objects in the drawing palette, such as a beta-distribution curve as a shape for a vapor dome, XDraw is customized to serve as the drawing interface for thermodynamics domain.

The overall architecture of Thermo Cycle Tutor includes eight components (Figure 4.2). They are 1) Web-based Drawing tool XDraw, as we mentioned before; 2) Domain model, which includes concepts and rules of the domain to be learned; 3) Expert module, which generates expert solution from domain rules; 4) Evaluation engine, which compares the student's drawing with expert's solution based on a series of check steps; 5) Pedagogical module, which defines pedagogical instructions based on student's activities; 6) Student model, which saves student's drawing and its corresponding tutor feedback for post analysis; 7) UI controller that controls displays and activities based on messages from tutor server; and 8) Training materials and pre/post test materials for assessment, provided by domain experts.



An inventor proposes the following cycle to accomplish refrigeration:

Figure 4.2. Thermo Cycle Tutor architecture.

Let's briefly describe a student's user experience with the tutor. After reading the problem description, the student is asked to draw a T-v diagram (see Figure 4.3) to describe the behavior of the system as the first step in framing the problem. The goal of the tutor is to assess students' decision making with regard to the drawing so that inferences can be made concerning their conceptual understanding of thermodynamics.

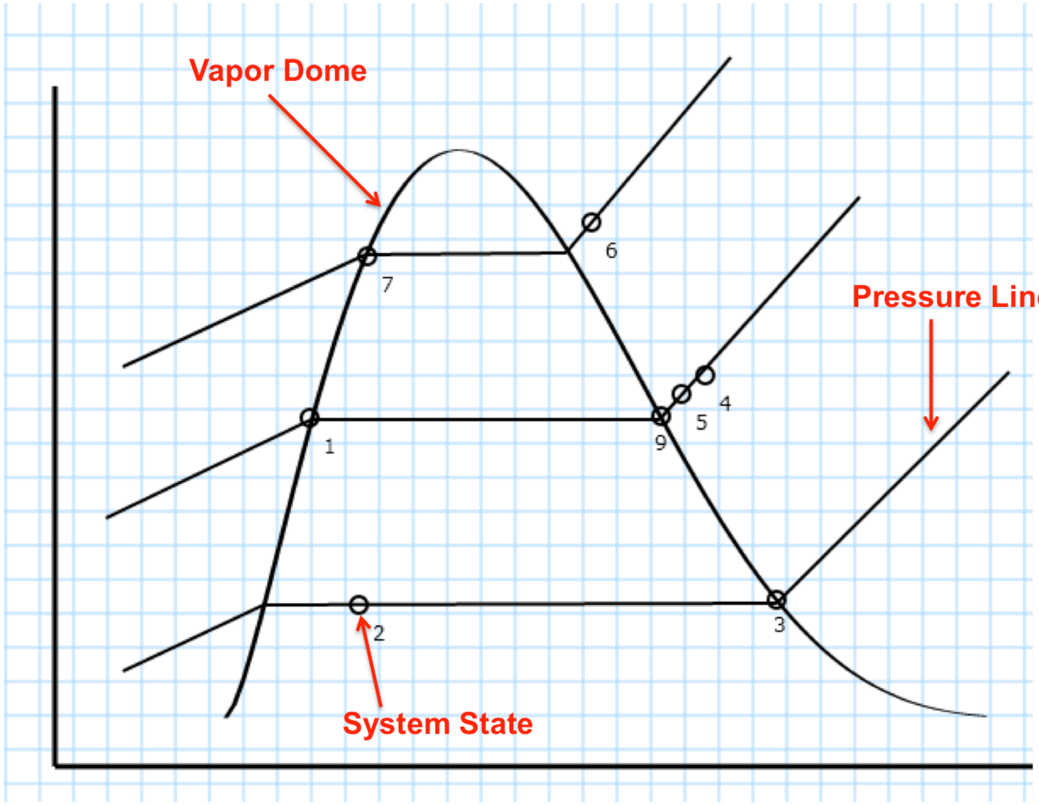


Figure 4.3. A T-v diagram example from the Thermo Cycle Tutor.

After the student submits the drawing, the UI controller saves the current version of the drawing and sends it to the evaluation engine through a TCP socket. The evaluation engine is based on domain model and expert module. The expert module is represented as a sequence of decisions that need to be made while describing the behavior of the system. Upon receipt of a request message from the UI controller, the evaluation engine assesses the drawing and sends a feedback message back to the UI controller. The feedback is encoded and saved with the current version of the drawing for each student. According to the feedback, student model (a state transition model) is updated to reflect the current state of the student's understanding. Once the student receives feedback, he/she will be redirected to the pedagogical module that may ask context-sensitive questions about concepts or display an instructional video. The student can then modify the diagram and re-submit. This process iterates until the student quits or successfully completes the diagram. In the following sections, we will discuss domain model, expert module, evaluation engine and student model in details.

4.3.1. Domain model

The tutor domain is focused on the physical properties of some commonly-used components in thermodynamics problems. Specifically, the goal is to understand how a system of components (e.g., a compressor) behaves in terms of pressure, temperature, specific volume, and entropy. Based on the physical properties, a rule is associated with each component. For example, when a fluid moves through a pump, a change in state occurs because the pressure will increase, while the temperature and specific volume will increase slightly. The domain author specifies the rules that are applied to the components (e.g. to assume constant specific volume). Table 4.1 shows the rules associated with system components. These rules are static and can be applied to many problems in the thermodynamics domain. Rules are saved in a database table and are retrieved to generate the expert solution when a new problem is authored. We will discuss the details in the expert module section.

Table 4.1. Rules for system component behavior. Note "between" is used to describe properties within component that has either two inputs and one output, or one input and two outputs.

Component	Pressure	Temperature	Specific Volume	Entropy
expansion valve	decreases	decreases	increases	increases
evaporator	same	same or increases	increases	increases
compressor	increases	increases	decreases	same or increases
mixing chamber	same	between	between	between
condenser	same	decreases or same	decreases	decrease
liquid -gas separator	same	same	between	between
pump	increases	same or increases	same, increase	same or increases
turbine	decreases	decreases	increases	same or increases

4.3.2. Expert module

The expert module is where expert knowledge represents in both domain-wide and problem-specific scales. It includes three parts: an expert solution, an evaluation sequence and feedback to student.

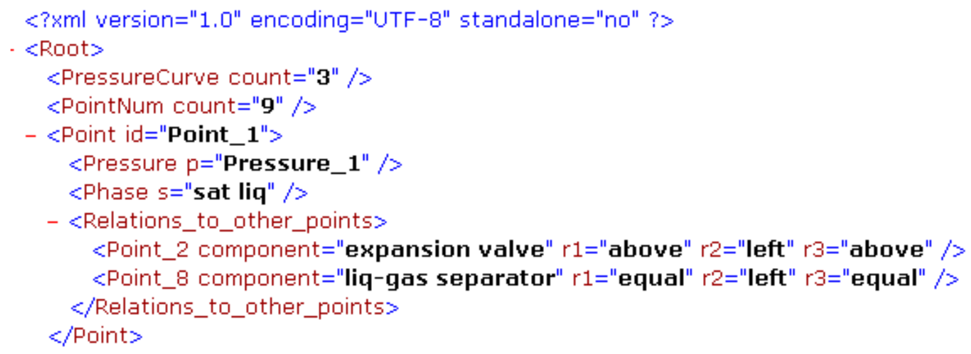
4.3.2.1. An expert solution to a given problem

Expert module uses domain rules to automatically generate an expert solution once problem-specific information is given. An expert solution is represented by an .xml file, which defines all elements in the solution and the relationship between elements. For the T-v diagram, the elements include a vapor dome, pressure lines, and states (locations and spatial relations to other states) from an expert's point of view as shown in Figure 4.4.

Step 1. Generate expert result table.

To generate the expert solution, system-input information is used to describe all the components in the system and the system states for a given problem. For each component, the input and output states are specified. An example of the ComponentState table for a mixing chamber having two input states and one output state is shown in Table

4.2. The SystemState table example shown in Table 4.3 describes the phase for each state. The phase information determines the location of a state in the T-v diagram. For example, if a state is in the saturated liquid phase, it should be located on the left edge of vapor dome, whereas if a state is in two-phase, its location should be underneath the vapor dome with no contact with the curve.



```
  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <Root>
    <PressureCurve count="3" />
    <PointNum count="9" />
  - <Point id="Point_1">
      <Pressure p="Pressure_1" />
      <Phase s="sat liq" />
    - <Relations_to_other_points>
        <Point_2 component="expansion valve" r1="above" r2="left" r3="above" />
        <Point_8 component="liq-gas separator" r1="equal" r2="left" r3="equal" />
      </Relations_to_other_points>
    </Point>
```

Figure 4.4. A snippet of an expert solution that was automatically generated from domain knowledge, ComponentState table and SystemState table.

Table 4.2. A ComponentState table example for a mixing chamber.

Component	Component ID	StateNo	StateType
mixing chamber	4	9	Input
mixing chamber	4	4	Input
mixing chamber	4	5	Output

Table 4.3. A SystemState table example.

State	Phase
1	saturated liquid
2	two-phase
3	saturated vapor
4	superheat
5	superheat
6	superheat
7	saturated liquid
8	two-phase
9	saturated vapor

An expert result table that describes system behavior is computed for a given problem based on the rules of the domain model, the ComponentState and SystemState table. In the expert result table shown in Table 4.4, "=" corresponds to no change, "<" corresponds to increase comparing to values when material flow enters the component, and ">" corresponds to decrease. The components in a specific system are matched with the behavior of pressure, temperature and specific volume. Some of the behaviors can have multiple results, such as the temperature change in an evaporator could increase or stay the same. So "<" or "="are both listed in the expert result table.

Table 4.4. An example of expert result table.

Component	Component ID	Input State	Output state	P	T	v
expansion valve	1	1	2	>	>	<
evaporator	2	2	3	=	=,<	<
compressor	3	3	4	<	<	>
mixing chamber	4	9	5	=	<,>,=	<,>,=
mixing chamber	4	4	5	=	<,>,=	<,>,=
compressor	5	5	6	<	<	>
condenser	6	6	7	=	=,>	>
expansion valve	7	7	8	>	>	<
liq-gas separator	8	8	1	=	=	<,>,=
liq-gas separator	8	8	9	=	=	<,>,=

Step 2. Determine the number of pressures and assign relative pressure to each state.

The expert module uses the expert result table to compute the total number of pressures in the given system based on pressure's equal and unequal relations. The algorithm first selects pairs of input state and output state that have equal sign in the P column in each row, and groups them together if they share either input state or output state. The number of distinct groups equals the number of pressures in the system. Then it checks the unequal relations between each group, and ranks them with a relative pressure value. States that belong to the same group would be assigned with a same relative pressure value.

Step 3. Compose the expert solution as an .xml file.

The final step is to convert the expert result table and relative pressure value to an .xml file. Figure 4.4 shows an excerpt of a final expert solution. It includes overall information about the drawing, such as the number of pressures, the number of points (i.e., states), and point-wise information as well. For each point, the file contains its id, relative pressure value, phase information and relationship to other related points. The related point is defined by the expert result table, where a pair of input state and output state within a component is regarded as related. Also, it translates the mathematics relation (“less than”, “equal” and “greater than”) in pressure, temperature and specific volume to spatial-wise relation, such as “left”, “right”, “above”, “below” or “equal”. For example, in T-v diagram, temperature and specific volume are represented in y and x axis, respectively. As State 1 has a larger temperature, but a smaller specific volume than State 2, it should be located on above and on the left side of State 2 in T-v diagram. Moreover, it has a larger pressure value, which means it should be located in a pressure line that is above State 2’s. As it is shown in Figure 4.4, r1, r2 and r3 represent temperature, specific volume and pressure relations, respectively.

4.3.2.2.Evaluation on student’s decisions

As we mentioned before, Thermo Cycle Tutor is a Decision-based Learning tutor, which implemented a set of decisions according to a thermodynamics expert Dr. Hagge who is also the co-author of this paper. Thus, the expert module includes a sequence of evaluation to understand student’s knowledge.

1). Is a vapor dome present? A student needs to determine that phase changes are involved and therefore, a vapor dome is necessary.

2) Does the diagram have the correct number of pressure lines? This is an indicator that the student understands system behavior. Pressure lines can be recognized if any three connected line segments are each in a separate region (liquid, two-phase and gas region), or two connected line segments with a horizontal segment in two-phase region and a segment in either liquid or gas region.

3) Are the pressure lines drawn correctly? This is another indicator that the student understands system behavior. Each line should have three segments, a positive slope segment in the liquid region, a horizontal segment in the two-phase region, and a positive slope segment in the superheat region.

4) Are the pressures for each pressure line and the phase change temperatures correctly identified? This indicates that the student knows how to find the correct pressure and temperature values. The pressure labels for each line should appear in the super heat region and the phase change temperature for each pressure needs to be identified on the temperature axis.

5) Are all the states included on the vapor dome? In problem description and state diagram, each state is given and labeled. Student is expected to use points to represent all

the states mentioned in problem statement, and locate each of them on an appropriate pressure line.

6). Are all the components correctly represented? Each component includes two or three states and is given in the state diagram. In domain model, we have discussed the general rules in each component. Student is expected to move around the state points so that the spatial relationship between states satisfies the physical and chemical properties in each component.

7). Is phase information of each state correctly identified? A vapor dome divides phase into five regions: liquid, saturated liquid, two-phase, saturated gas and super-heat. The last step is to check if a state is at the correct region. Student is expected to move around the points to the correct position. As you may notice, step (6) and (7) are correlated. Moving a point to another position might change the relative location to some other points. In order to prevent student falling into the "error-in-a-loop", tutor wants to ensure student knows the relative location of each point before he/she anchors the point to a region.

4.3.2.3. Feedback to student

Expert module also contains feedback to students for each check step. The feedback could be explicit, such as "State 1 should be saturated liquid at middle pressure", if student fails to put State 1 at left edge of vapor dome. Or it could be implicit by forwarding them to some other activities. For example, if the number of pressures is incorrect, tutor's feedback could be "There appears to be some misconceptions about the number of pressures in a refrigeration cycle. Let's examine the individual components in the cycle and how they affect the pressure." To make it effective and easy-to-understand, we consulted a domain expert to compose the feedback. The feedback was prepared for both pass and fail in each check monitored by the evaluation engine.

4.3.3. Evaluation engine

Evaluation engine is where the tutor evaluates student's drawing and sends result message to the UI controller. When a student clicks the *submit_drawing* button, the UI controller immediately sends the drawing file to evaluation engine. Once the evaluation engine receives the file, it converts the drawing to a .xml file with similar structure as the expert solution, by detecting information such as pressure lines, state location and so on. Then it compares the generated file with expert solution by following the check procedure defined by the expert module, and sends feedback message to UI controller when either error is found or the student successfully passes the problem.

In general, it contains four parts: 1) A module to create a structured .xml document based on student's drawing; 2) A pool of check functions that can be called by different applications; 3) A start model to initialize and begin check process; 4) Feedback to UI controller.

4.3.3.1. A module to create a structured .xml document

The structured document has the same elements as the expert solution, such as number of pressures, number of states, and for each state, the relative pressure, phase information and relationship to other states. To make it happen, there are some challenges.

a). Handle different shapes of vapor dome input.

The shape and location of a vapor dome is crucial in determining pressure lines and understanding the relative location of points. To draw a vapor dome, a student first selects a vapor dome template which follows a beta distribution from the tool bar, places it at any location in the canvas, and then drags it both horizontally and vertically to reach a certain size. Two steps were carried out to handle varied shapes and origin locations: 1) Drawing's origin is adjusted based on vapor dome's origin. 2) An inverse function is used to map x and y values in the vapor dome back to a template shape. By this means, tutor is able to interpret a vapor dome object in varied shapes and locations.

b). Handle sloppy drawings of pressure line.

In a T-v diagram in general, a complete and correct pressure line should contain three segments: a horizontal segment in two-phase region, two segments with each in liquid and gas region with a positive slope, respectively. However, student might produce varied patterns based on his/her procedure knowledge. (I.e. how to represent a pressure line.) One of the common mistakes is, instead of including three segments, student tends to draw two segments, a horizontal line in the two-phase region and a sloped line in either liquid or gas region. Or instead of drawing a positive-slope segment in liquid or gas region, a very low-slope (<10 degrees) line is drawn. The procedure knowledge is different from conceptual knowledge. I.e., student knows the possible number of pressures in the system, but fails to draw the pressure line correctly. To handle this, tutor uses larger tolerance when performing the initial checks, but decreases the tolerance as the student proceeds.

c). Handle some sophisticated situations.

Sometimes, when a student starts to frame the problem by sketching a few number of pressure lines, he/she doesn't have a clear understanding of how many lines should be there. After anchoring all the states on appropriate pressure lines, he/she realizes only some of the pressure lines are valid, but leaves the extra ones empty. Evaluation engine should be able to recognize the empty pressure lines and doesn't count them when interpreting the number of pressures. By this means, it gives students more freedom and encourages them to explore the problem space as much as possible before going to the details. Thus, Thermo Cycle Tutor is able to handle student's action with a more accurate understanding.

4.3.3.2. A pool of check functions

Basically, check functions support the comparison of the expert solution with student's understanding. The pool contains several routines that can be called by any application. For example, to check if a student includes all the states, a check function is called by first retrieving the number of state values in expert solution and student's drawing, and then compare the two values. If student's solution has less number of states, the function will return the missing state id, so that the UI controller could take further actions. To make them modular and easy to use, we design some functions that can be reused by taking state id and attribute as inputs. For example, to check how pressure, temperature and specific volume changes in expansion valve, only one function is associated. This function would take state id 1 and 2 that represent expansion valve in a problem set, plus the name of one property, such as pressure. To check the other properties, the function can be re-called by the evaluation process. In summary, the idea of check functions is to allow evaluation step be reused and re-organized by different applications. In the future, it will become a support for the authoring tools, where domain experts can issue the type and order of evaluation by simply selecting and grouping the check functions.

4.3.3.3. A start model

A start model is built on top of the check functions. It serves as a problem-specific agent of the expert module. Basically, it maps the evaluation steps defined by expert module to check functions. Also, it attaches feedback from expert module to the mapped check functions. For example, to complete the component check, which is the 6th step defined by expert module, the start model would first break it down into 8 components check. And then for each component, it calls the underlying check functions by taking the input and output state ids, as well as the attribute name (P, T or v). It is noteworthy, expert solution might not include all the information in the problem space, i.e. the exact phase change temperature under a certain pressure. Instead, it contains the overall picture of relative pressure value and relative spatial relation information for each component. Thus, in order to support numeric value check, the start model needs to take specified value from user input and calls the relevant check function. For example, to check pressure label, expert must tell the evaluation engine the correct value and tolerance he/she could accept, since such information is not represented by expert solution. The underlying check functions are located in the same function pool as the other functions. The only difference is that the correct answer is retrieved from user input instead of from the generated expert solution. This add-on model gives tutor more flexibility to handle both drawing's spatial relation and numeric value input.

4.3.3.4. Feedback to UI controller

There are two types of feedback to UI controller. One is the feedback message that would be displayed to student. This type of feedback is specified by the expert module initially, and mapped by start model as evaluation process starts. Another one is the action feedback to the UI controller. The action feedback is designed to support further pedagogical-oriented tutorials or activities. Furthermore, in order to facilitate tracking students' performance and analyzing their changes of performance and understanding, evaluation engine encodes pass or fail results from each check step. The codes include results from all the check steps defined in the evaluation procedure, even though feedback message only targets on the first encountered error. The codes are saved every time a new request is submitted, which gives tutor a better understanding of student's current state.

4.3.4. Pedagogical module

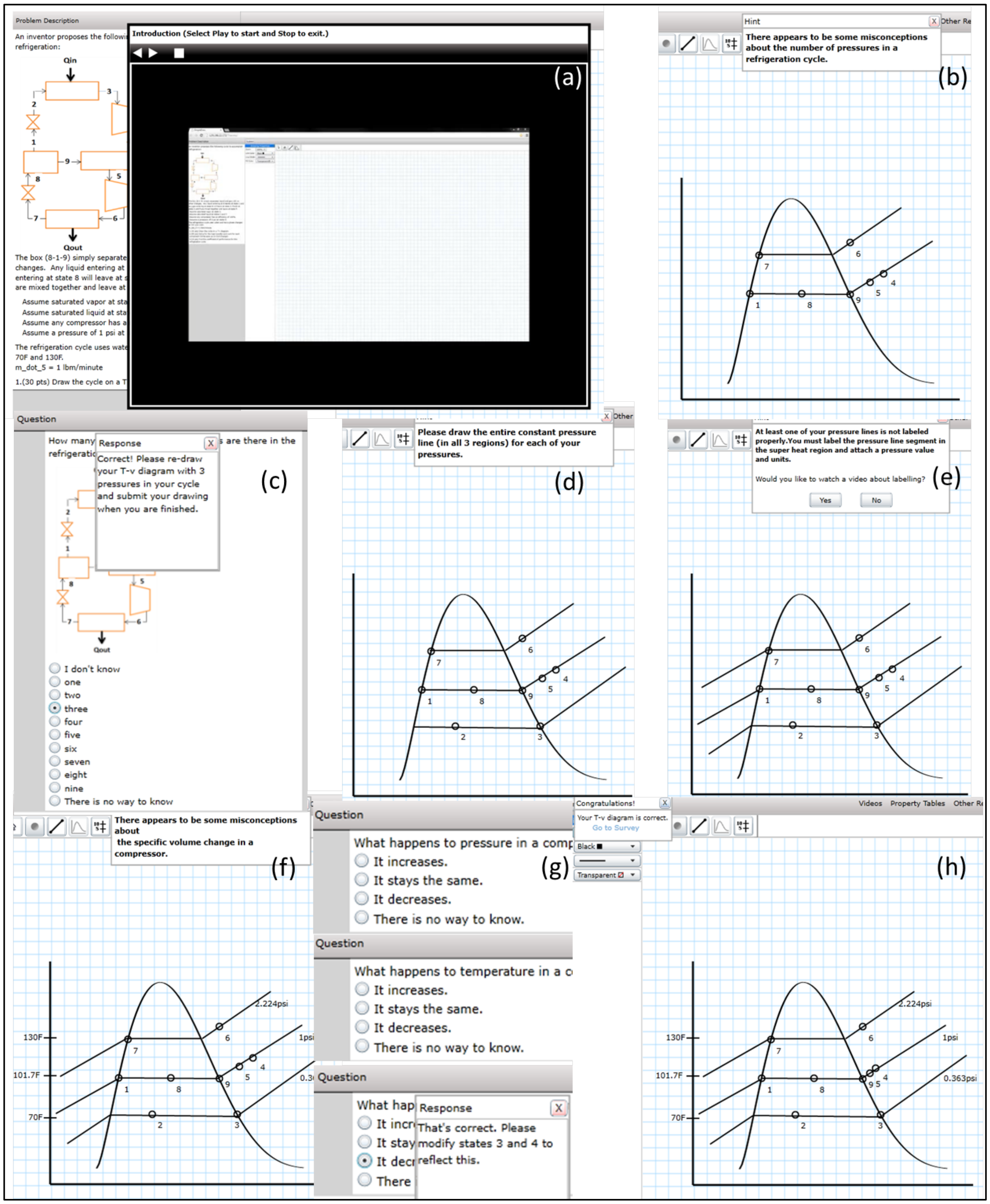
A pedagogical module defines what pedagogical instructions should be provided based on students' actions. This module was developed based on pedagogies from one thermodynamics expert Dr. Hagge. In general, it includes 1) the types of feedback available on tutor server, such as texts in a pop-up window, erred drawing piece with highlighted color, standalone tutorial videos, multiple-choice questions and pictorial illustration, and 2) the mapping between the types of feedback to student's error or misconception. Details of this module can be found in his separate paper on Decision-based Learning Tutor (Hagge et al., submitted).

4.3.5. Student model

A student model is constructed for the purpose of tracking students' performance, recording their misunderstandings and facilitating instructor's further analysis. Generally speaking, it contains information from three resources. 1. Student's drawing data from the drawing interface. The drawing is automatically saved once student sends a request for hint. The drawing data is an .xml file that contains all the objects' spatial information as well as their text labels. It is saved along with student's id and a timestamp. Student's drawing data can be retrieved by querying the user id for post analysis. 2. Feedback message and evaluation code generated by the evaluation engine. As we mentioned before, feedback conveys information either a hint if an error is found or a confirmation if the drawing passes the check. When a hint button is clicked, a feedback would be displayed in a pop-up window, and be saved in the student model as well. 3. Survey, pre-test and post-test data from a third-party software. To better understand students' background, their domain knowledge and other subjective information, a survey is prepared before and after the students use Thermo Cycle Tutor. Pre-test and post-test are used for the purpose of evaluating students' knowledge gains. By having these data, tutor could have a deeper understanding of the students' knowledge state, their knowledge gains after using the tutor, how they feel the effectiveness of the tutor, and possible suggestions on the features of the software.

4.4.Tutor Session

In this session, we will give an example to show how tutor helps a student to go through a refrigeration cycle problem. As is shown in Figure 4.5 (a), when the tutor was launched, an introduction video about how to use the tutor was displayed at the central of the canvas. After exiting it, the student drew a vapor dome and two sets of connected lines to represent pressure lines. He also added some points on the pressure lines to indicate the states. When *submit_drawing* button was clicked, a feedback was displayed, "There appears to be some misconception about the number of pressures in a refrigeration cycle." A multiple-choice question was popped up to clarify if student has a misconception of the number of pressures by asking "how many pressures are there in the refrigeration cycle?” As student correctly answered the question, tutor confirmed it and prompted him to re-draw the diagram by applying 3 pressures. After he added one more pressure line and clicked the submit button, a feedback popped up “Please draw the entire constant pressure line in all 3 regions for each of your pressures”. Student was told the pressure line was incomplete. So he added a positive-slope line on liquid region for each pressure line, and hit feedback. He then was expected to label pressure lines in super-heat region, and an option to watch a demo video about how-to-label. Pressure value for a given temperature can be found in property tables in right upper corner. When the student labeled the pressure value and phase change temperature accordingly, and clicked the *submit_button* again, he got a message saying, “There appears to be some misconceptions about the specific volume change in a compressor.” He was then directed to three multiple-choice questions about the changes of pressure, temperature and specific volume in a compressor. As he successfully answered these questions, a prompt message said, “Please modify state 3 and 4 to reflect this.” He then squeezed state 9, 5 and 4, so that state 4 can be drawn on the left side of state 3. After the modification, he resubmitted the drawing and got a confirmation message saying his drawing was correct.



Problem Description

An inventor proposes the followi
refrigeration:

The box (8-1-9) simply separate
changes. Any liquid entering at
entering at state 8 will leave at s
are mixed together and leave at

Assume saturated vapor at sta
Assume saturated liquid at sta
Assume any compressor has a
Assume a pressure of 1 psi at

The refrigeration cycle uses wate
70F and 130F.
m_dot_5 = 1 lbm/minute

1.(30 pts) Draw the cycle on a T

Introduction (Select Play to start and Stop to exit.)

Hint

There appears to be some misconceptions about the number of pressures in a refrigeration cycle.

Question

How many … s are there in the refrigeratio

Response

Correct! Please re-draw your T-v diagram with 3 pressures in your cycle and submit your drawing when you are finished.

- I don't know
- one
- two
- [✓ three]
- four
- five
- six
- seven
- eight
- nine
- There is no way to know

Please draw the entire constant pressure line (in all 3 regions) for each of your pressures.

At least one of your pressure lines is not labeled properly.You must label the pressure line segment in the super heat region and attach a pressure value and units.

Would you like to watch a video about labelling? Yes No

There appears to be some misconceptions about the specific volume change in a compressor.

Question

What happens to pressure in a comp
- It increases.
- It stays the same.
- It decreases.
- There is no way to know.

Question

What happens to temperature in a c
- It increases.
- It stays the same.
- It decreases.
- There is no way to know.

Question

What hap
- It incr
- It stay
- [✓ It decr]
- There

Response

That's correct. Please modify states 3 and 4 to reflect this.

Congratulations!

Your T-v diagram is correct.

Go to Survey

Black
Transparent

Figure 4.5. An example showing student's interaction with Thermo Cycle Tutor.

4.5. Tutor Evaluation

Thermo Cycle Tutor has been tested by 7 groups of engineering undergraduates with a total number of 373 at Iowa State University in last two years. To evaluate tutor's effectiveness, a pre-test containing 18 multiple-choice questions about the physical properties of 6 components was provided before students used the tutor. After about 40 minutes activity with the tutor, students were asked to answer the same questions. A paired t-test on the test scores was performed. Every group of students' test has shown a large improvement in understanding, regardless of the instructors or class size (Figure 4.6). Detailed discussion will be included in a separate publication by Dr. Hagge.

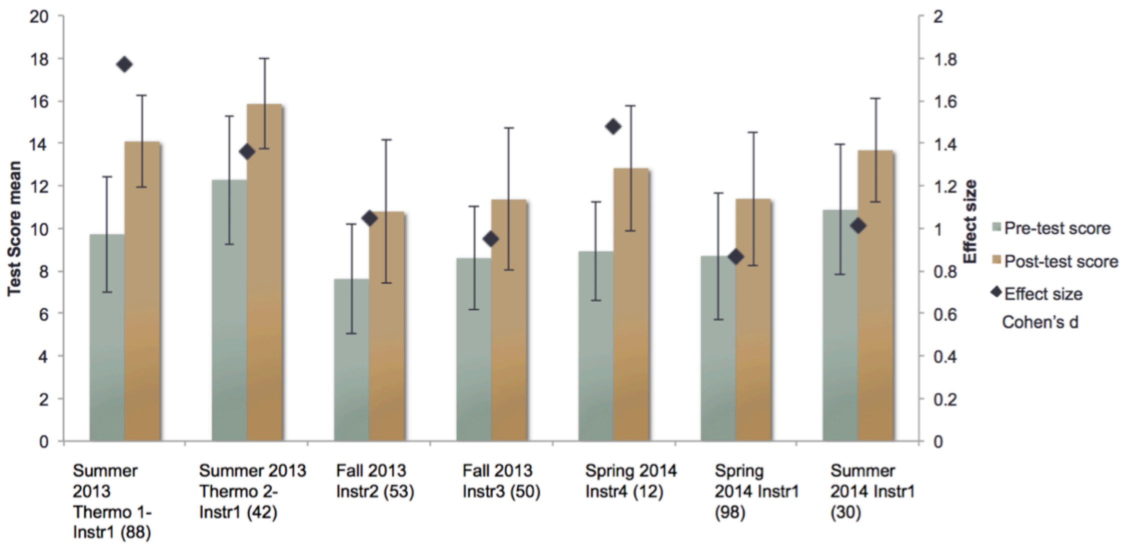


Figure 4.6. Tutor test results of 373 engineering undergraduates. Numbers inside the bracket show the number of students in each group. From left to right, *p*-values for paired t-test: 7.5E-21, 3.0E-10, 7.7E-12, 7.9E-7, 4.9E-5, 2.33E-11, and 3.47E-18. Cohen's d is calculated as the effect size, shown as black diamonds. Note: Summer 2013 Thermo 1 students' score is out of 16 possible points due to poorly worded pre/post questions. The rest of the scores are out of 18 possible points.

4.6. Conclusion and Future Work

In this paper, we present Thermo Cycle Tutor, a Decision-based Learning tutoring system to help engineering undergraduates in thermodynamics courses. In general, it contains eight components: drawing interface, domain model, expert module, evaluation engine, pedagogical module, student model, UI controller, and training materials. Drawing interface allows students to pick and draw objects on a canvas and seek help from tutor. Expert solution can be generated automatically in expert module once problem-specific information is given. Evaluation is taken place in evaluation engine where a set of decisions has to be met in order to pass. The system implemented pedagogical instructions from a thermodynamics expert. Furthermore, UI controller controls the data flow and communication message between drawing interface and evaluation engine.

As we discussed before, a student model is used to store student's drawing information as well as his/her knowledge states. Currently, the student model only supports post analysis by instructors. In the future, we'd like to incorporate it with evaluation engine in real-time feedback composition. If a student gets the same mistake for 3 times (error-in-a-loop), instead of repeating the same feedback message, student model should alert the evaluation engine that a special feedback is needed to help the student jump out of the loop. Also, if some inconsistency between the pre-test answer and the drawing is found, student model should inform the evaluation engine to give a more personalized feedback. For example, the pre-test results show that the student has a correct understanding of how pressure changes in expansion valve. However, the drawing doesn't represent the correct relationship between input and output state in an expansion valve. Obviously, the mistake doesn't come from misconceptions. Instead, it might come from incomplete procedure knowledge. (I.e., how to map the physical and chemical relations within a component into a spatial relation between states.) Therefore, instead of sending him/her for activities that correct the conceptual misunderstanding, tutor might give a tutorial on how-to-represent-states on vapor dome.

Also, student model could be used to guide evaluation engine for different check options. As we know, the problem-specific checks defined in expert module is a complete map of the path that a student should go through in order to solve the problem. It contains the knowledge space that a human expert expects a successful problem solver. The main part of the tutor is to convert that knowledge to some measurable actions, such as drawing of states or pressure line, and then evaluate their outcomes. However, some of the students might have stronger background or have prior experience in solving similar problems. When framing the problem, those students may skip some of the steps and reach the final outcome in a shorter path. For instance, after the student identifies there are 3 pressure lines and draws them correctly, he/she can anchor states in appropriate lines directly. Identifying the pressure value and phase change temperature would be helpful to understand the problem and be beneficial for future problem solving. It is not counted as a necessary step, however. To benefit both advanced students and average students, the evaluation engine could consult student model to classify if the student needs a complete check path or only a few key checks. The goal of constructing student model is not only to understand students' background knowledge, interpret their misconception, support personalized feedback and post analysis, but also provide each student an individualized tutoring experience.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant Number EEC-1025133. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Anderson, E. E., & Taraban, R. (2005). Implementing and Assessing Computer- based Active Learning Materials in Introductory Thermodynamics. *International Journal of Engineering Education*, *21*(6), 1168–1176.

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist*, *51*, 355–365.

Baghaei, N., & Mitrovic, A. (2005). COLLECT-UML: Supporting Individual and Collaborative Learning of UML Class Diagrams in a Constraint-based Intelligent Tutoring System. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part IV* (pp. 458–464). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/11554028_64

Beall, H. (1994). Probing Student Misconceptions in Thermodynamics with In-Class Writing. *Journal of Chemical Education*, *71*(12), 1056. doi:10.1021/ed071p1056

Bursic, K. M. (2012). Does The Use Of Clickers Increase Conceptual Understanding In The Engineering Economy Classroom? In *Proceedings of the 2012 ASEE Annual Conference*. San Antonio, TX.

Chen, J. C., Whittinghill, D.C., Kadlowec, J. A. (2010). Classes That Click: Fast, Rich Feedback to Enhance Student Learning and Satisfaction. *Journal of Engineering Education*, *99*(2), 159–168.

Chi, M. T. H. (2005). Commonsense Conceptions of Emergent Processes: Why Some Misconceptions Are Robust. *Journal of the Learning Sciences*, *14*, 161–199.

Corbett, A. T., Koedinger, K. R., & Hadley, W. S. (2002). Cognitive Tutors: From the research classroom to all classrooms. In P. S. Goodman (Ed.), *Technology enhanced learning: Opportunities for change* (pp. 235–263). Mahway, NJ: Lawrence Erlbaum Associates.

Diefes-Dux, H. A., Salim, A. (2009). Problem Formulation during Model-Eliciting Activities: Characterization of First-Year Students' Responses. In *Proceedings of* *the Research in Engineering Education Symposium 2009*. Palm Cove, Queensland, Australia.

Dolenc, K., & Aberšek, B. (2015). TECH8 Intelligent and adaptive e-learning system: Integration into Technology and Science classrooms in lower secondary schools. *Computers & Education*, *82*, 354–365. doi:10.1016/j.compedu.2014.12.010

Education, U. S. D. of. (2013). What Works Clearinghouse. High School Mathematics intervention report: Carnegie Learning Curricula and Cognitive Tutor®. Retrieved from http://whatworks.ed.gov.

Engineering, N. A. of. (2008). NAE grand challenges for engineering. Retrieved from http://www.engineeringchallenges.org/cms/8996/9127.aspx.

Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2011). CogSketch: Sketch Understanding for Cognitive Science Research and for Education. *Topics in Cognitive Science*, *3*(4), 648–666. doi:10.1111/j.1756-8765.2011.01149.x

Guo, E., Gilbert, S., Jackman, J., Starns, G., Hagge, M., Faidley, L., & Amin-Naseri, M. (2014). StaticsTutor: Free Body Diagram Tutor for Problem Framing. In *Proceedings of the 12th International Conference of Intelligent Tutoring Systems.* (Vol. 8474, pp. 448–455). Honolulu, HI.

Hagge, M., Amin-Naseri, M., Jackman, J., Guo, E., Gilbert, S., Starns, G., Faidley,L. (submitted to *Advances in Engineering Education*). Decision-based Learning for Thermodynamics Phase Diagram.

Hattie, J., & Timperley, H. (2007). The Power of Feedback. *Review of Educational Research*, *77*(1), 81–112. doi:10.3102/003465430298487

Lovett, M., Meyer, O., Thille, C. (2008). The Open Learning Initiative: Measuring the effectiveness of the OLI statistics course in accelerating student learning. *Journal of Interactive Media in Education.*

Miller, R. L., Streveler, R. A., Olds, B. M., Chi, M. T. H., Nelson, M. A., & Geist, M. R. (2006). Misconceptions about rate processes: preliminary evidence for the importance of emergent conceptual schemas in thermal and transport sciences. In *Proceedings of the American Society for Engineering Education Annual Conference*. Chicago, IL.

Oliveira Neto, J. D. De, & Nascimento, E. V. (2012). Intelligent Tutoring System for Distance Education. *Journal of Information Systems and Technology Management*, *9*(1), 109–122. doi:10.4301/S1807-17752012000100006

Patterson, B., Kilpatrick, J., Woebkenberg, E. (2010). Evidence for Teaching Practice: The Impact of Clickers in a Large Classroom Environment. *Nurse Education Today*, *30*(7), 603–607.

Perkins, K. K., Turpen, C. (2009). Student Perspectives on Using Clickers in Upper-division Physics Courses. In *2009 Physics Education Research Conference. AIP Conference Proceedings* (pp. 225–228).

Prevost, L., Haudek, K., Urban-Lurain, M., & Merrill, J. (2012). Examining student constructed explanations of thermodynamics using lexical analysis. In *Proceedings of the 2012 Frontiers in Education Conference*. Seattle, WA.

Rai, D., & Beck, J. E. (2012). Math Learning Environment with Game-Like Elements : An Incremental Approach for Enhancing Student Engagement and Learning Effectiveness. *Proceedings of the 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012.*, 90–100.

Redish, E. F., Smith, K. A. (2008). Looking beyond content: Skill development for engineers. *Journal of Engineering Education*, *97*(3), 295–307.

Reiner, M., Slotta, J. D., Chi, M. T. H., & Resnick, L. B. (2000). Naive Physics Reasoning: A Commitment to Substance-Based Conceptions. *Cognition and Instruction*, *18*, 1–34.

Roselli, R. J., & Howard, L. (2003). Integration of an Interactive Free Body Diagram Assistant with a Courseware Authoring Package and an Experimental Learning Management System. In *Proceedings of the American Society for Engineering Education Annual Conference*. Nashville, TN.

Schiaffino, S., Garcia, P., & Amandi, A. (2008). eTeacher: Providing personalized assistance to e-learning students. *Computers & Education*, *51*(4), 1744–1754. doi:10.1016/j.compedu.2008.05.008

Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, *78*(1), 153–189.

Sklavakis, D., & Refanidis, I. (2013). MATHESIS : An Intelligent Web-Based Algebra Tutoring. *International Journal of Artificial Intelligence in Education*, *22*, 191–218. doi:10.3233/JAI-130036

Steif, P. S., A. Dollár, A. (2009). Web-based Statics Course: Study of Usage Patterns and Learning Gains. *Journal of Engineering Education*, *98*(321-333).

Streveler, R. A., Litzinger, T. A., Miller, R. L., & Steif, P. S. (2008). Learning conceptual knowledge in the engineering sciences: Overview and future research directions. *Journal of Engineering Education*, *97*(3), 279–294.

Taraban, R., Anderson, E. E., Sharma, M. P., & Weigold, A. (2003). Developing a model of students' navigations in computer modules for introductory thermodynamics. In *Proceedings of American Society for Engineering Education Annual Conference*. Nashville, TN.

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *Int. J. Artif. Intell. Ed.*, *15*(3), 147–204. Retrieved from http://dl.acm.org/citation.cfm?id=1434930.1434932

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational approaches to the communication of knowledge*. Morgan Kaufmann Pub.

Zakharov, K., Mitrovic, A., & Ohlsson, S. (2005). Feedback Micro-engineering in EER-Tutor. In *Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning Through Intelligent and Socially Informed Technology* (pp. 718–725). Amsterdam, The Netherlands, The Netherlands: IOS Press. Retrieved from http://dl.acm.org/citation.cfm?id=1562524.1562620

CHAPTER 5

INSTRUCTIONAL STRATEGIES IN DIAGRAM-BASED ITSS: LESSONS LEARNED FROM TWO TUTORING SYSTEMS

This chapter is an extended version from a paper published in *the 12th International Conference on Intelligent Tutoring Systems.*

Enruo Guo, Stephen Gilbert

Abstract

Unlike text-based input in an Intelligent Tutoring System (ITS), a diagram is perceived as a whole state; the operation sequence is less important. Traditional step-wise coaching is not as appropriate in Diagram-based ITSs. From two previous tutoring systems, StaticsTutor and Thermo Cycle Tutor, we propose cross-domain pedagogical guidelines for Diagram-based ITSs. In particular, instruction needs to be mapped to a hierarchical understanding of the diagram, where each level focuses on different characteristics of the drawing. Also, a personalized evaluation sequence is desired, where evaluation steps are ordered by the student's current knowledge. Finally, instruction needs to address conceptual knowledge and procedure expertise separately. Some practical suggestions are described to achieve these goals, such as 1) use different tolerances for error at different levels of evaluation, 2) use Q&A to resolve diagram ambiguity, 3) early loading of expertise that is important to avoid difficult-to-fix diagrammatic states and 4) determine if the new evidence violates the existing understanding of the student.

5.1. Introduction

Given the comprehensive advantages of pictorial representations, diagrams play a big role in scientific cognition, e.g., free-body diagrams in physics, T-v diagrams in thermodynamics, circuit diagrams in electrical engineering, and UML diagrams in software engineering. Cognitive models of graphics comprehension (Shah, 1997) propose that graphics comprehension involves interaction between bottom-up perceptual processes of encoding information from the graphic as well as top-down processes of applying graph schemas and domain knowledge, which makes it a challenge to teach students how to use diagrams to represent information.

In this paper, we discuss lessons regarding pedagogy that we learned from two Diagram-based ITSs, and provide cross-domain guidelines for the design of future Diagram-based ITSs.

5.2. Background and Previous Work

We have designed and implemented two Diagram-based ITSs in engineering statics and thermodynamics courses: StaticsTutor (Guo et al., 2014) for free-body diagrams, and Thermo Cycle Tutor (Guo et al., 2015), for T-v diagrams in refrigeration cycles. Even though they focus on different domains, pedagogy in both systems aimed at helping students' conceptual understanding and decision-making at the earliest stage of problem framing.

StaticsTutor was developed to analyze student's drawn free-body diagrams and recognize misconceptions without requiring numerical force values or the need to provide equilibrium equations. Preliminary results with 81 engineering undergraduates in fall 2013 showed that StaticsTutor could detect students' misconceptions that were categorized as “missing basics,” “hinge issue,” “rope issue,” and so on. A post-survey indicated an overall positive experience with the tutor with a mean usability score of 3.5 (SD 1.11).

The Thermo Cycle Tutor implemented a teaching pedagogy (Hagge et al., submitted) based on decision-making, where class concepts are posed as a set of simple questions that can be answered for all problems in the thermodynamics courses. In fall 2013, 42 undergraduate engineering students were given a pre-test on refrigeration cycles and then given the Thermo Cycle Tutor to complete a homework problem. They then took a post-test. Students’ post-test scores improved from 70% to 89% on average. To test retention, they were given a second post-test after four weeks, and they scored an average of 81% better than the pre-test with no additional lectures on refrigeration cycles.

Both tutors faced the challenge of how to analyze the students' diagrams computationally, and how to give appropriate feedback. Pedagogical questions that arise include, “If there are multiple issues with the diagram, which issue should receive feed-back first?” “Given an error in the diagram, what can I infer about the student's misconceptions, if any?” “When should I evaluate the diagram, at each step of construction, or only at the end?”

5.2.1. Previous work on diagram interpretation

Koffman and Friedman (1976) designed an early instructional tool for diagramming to assist beginning programmers in learning to make a computer-aided flow diagrams. They emphasized the problem-framing aspects of diagram planning, and wanted students to use the diagrams to learn the program logic before implementing the code. Because these diagrams represent the flow of a program's state, they can be represented mostly linearly, with some loops. As students added the flow diagram in the authors' system, each new component was evaluated for whether it fit with the existing logic, and feedback was given if there was a conflict.

Usually it is difficult to analyze a diagram at each step of its construction, because there are typically graphic elements that must be added one at a time (though in no particular order), and at some points during this construction process, the diagram does not make sense if viewed in its partially completed state. However, in Koffman and Friedman's case, the linear structure and the level of granularity of their diagram components helped this system avoid these open-ended ambiguities that usually occur during construction.

Futrelle (1990) attempted to apply a level of abstraction to diagrams by offering a diagram grammar and process for automatic computational diagram analysis loosely based on computer vision. His approach, however, was focused on analyzing the diagrams, rather than tutoring using diagrams.

5.2.2. Pedagogy in ITSs

A class diagram editor for UML was developed by Py et al. (2008) that focused on the learner's metacognitive skills in the modeling task. They designed three types of interventions: 1) Notify, general feedback drawing student's attention to some part of the diagram; 2) Question, asking the learner about the diagram's properties; and 3) Propose, very specific feedback suggesting how to correct the diagram. In the present research, the focus is how to match the feedback to the appropriate hierarchical level of abstraction within the diagram. Perhaps future work might combine these approaches.

It is worth mentioning pedagogy in Andes tutor (Conati, Gertner, VanLehn, & Druzdzel, 1997). Instead of limiting students to a predefined optimal solution, it allows them to explore the problem solving space. A solution graph representation, which contains several types of nodes, is used to model all possible solution paths, upon which a Bayesian network is built. Then Bayesian inference is applied to designate student's current goal node and a rule-application node where the student is stuck in the middle. A hint is then generated to coach that knowledge accordingly. Even though Andes focuses on text-based inputs, this step-by-step coaching strategy also applies to diagram-based systems. However, there are some differences that make pedagogy in diagrams challenging: 1) A diagram should be perceived as an entire state, no matter when and how an element is added to the diagram. Step-by-step coaching needs to be redesigned appropriately. 2) Even though sequence is less important in a diagram, it does require a series of actions to be applied in order to meet a certain requirement in a given state. This means that the diagram must be properly defined as several sequential stages, where each stage represents certain conceptual understanding. Within a stage, the sequence of actions does not likely matter.

5.3. Instructional Guidelines in Diagram-based ITSs

5.3.1 Instruction needs follow a hierarchical understanding of the diagram

Even though diagrams vary across domains, there are usually underlying concepts that drive core questions that should be answered during the assessment process. The core questions can be defined through an expert module, which might vary based on the expert's instructional and pedagogical preferences. However, a general architecture that fits in a cross-domain evaluation system is highly desired. For this purpose, we propose three levels of hierarchy for diagram evaluation. Before defining the levels theoretically, we offer an illustrative example from thermodynamics. Figure 5.1 shows three T-v diagrams. These diagrams are used to abstractly represent how pressures, temperature, and volumes change within a mechanical refrigeration system, which may contain compressors, pumps, valves, etc. The Thermo Cycle Tutor basically asked six questions:

1. Is a vapor dome needed? (Are there phase changes in the cycle?)
2. How many pressures are present in the cycle?
3. How is a pressure line drawn on a T-v diagram?
4. How should phase change P and T be labeled on the diagram?
5. What are the P, T and v relations for each component?
6. How can the problem information, and the decisions above uniquely identify each state?

While these questions are particular to refrigeration cycles, they have the following characteristics which apply across domains: some of them focus on the student's conceptual understanding (1, 2, 5), and some focus on the procedural skill of how to make a diagram appropriately (3, 4). Of course, these two aspects are tightly coupled, and some questions apply to both (6).

It is noteworthy that the six questions follow a hierarchical understanding of the diagram. At Level 1, nine straight-line segments are recognized on a vapor dome (Figure 5.1a), where each three connected segments represent a pressure line. At this level, the message that the diagram conveys is simply that there are three pressures in this system. At Level 2 (Figure 5.1b), more details are shown: some text labels are attached to the pressure line segments at the right-hand side, and tick marks are added to show the phase change temperatures. These additions give the viewer more concrete information about the exact value of the pressures and phase change temperatures. Then, at Level 3 (Figure 5.1c), by adding some points with labels on the pressure line segments, the diagram brings in details on the state information and how it interacts with the pressures and phase change temperatures.

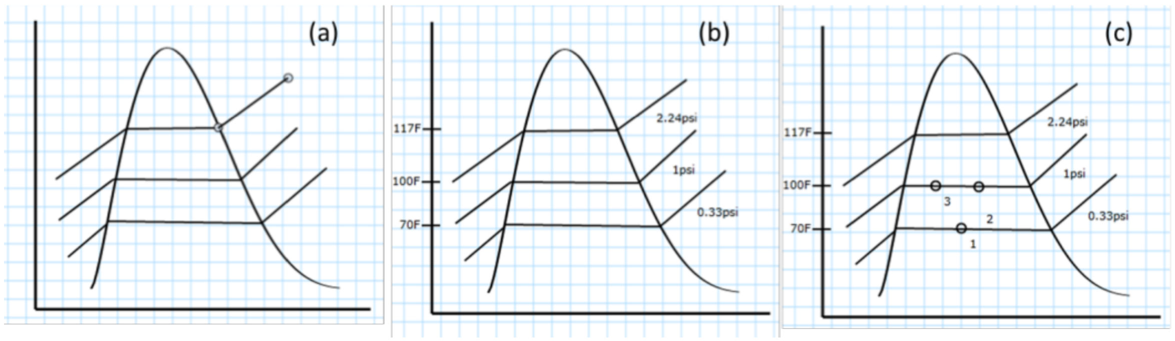


Figure 5.1. Three levels in a refrigeration cycle T-v diagram. (a). A vapor dome with three pressures. (b). Labels of phase-change temperature and pressure values were added. (c). State information was anchored on the pressure line.

To generalize the levels just described, Level 1 focuses on basic graph-style structures and the spatial relations between each other. At Level 1, the tutor has a rough idea of what components are present and their connections. To give feedback at Level 1, the Diagram-based ITS needs to incorporate domain knowledge. Level 2 focuses on object identities and their *object-type-specific* relationships. At this level, attributes about an object will be identified through domain knowledge and some text labels. These include object name and possible values relate to object. Spatial relationships from Level 1 will be transformed to more specific numerical relationships. As is shown at Figure 5.1 (b), the numeric value has been explicitly shown in each pressure line, so it is easy to tell the second pressure is 0.67 (1 - 0.33) psi higher than the first pressure, whereas Figure 5.1 (a) only tells the second pressure is above the first one. Level 3 focuses on properties or children of Level 2 objects. In this level, details on Level 2 objects will be revealed and examined. The details could comprise sub-objects that constitute a Level 2 object, or sub-objects that are attached to a Level 2 object but themselves are not considered as basic structures at Level 1.

Instructional feedback can be composed based on the level of specificity. The lower level error should be tackled first, as it is more fundamental and serves as the basis of the higher-level object. For instance, if a Level 1 object is missing, it doesn't make sense to correct a Level 2 object as by definition its structure is based on Level 1 object. We propose this "divide-and-conquer" strategy where each piece can be mapped to one or more states of student's understanding.

5.3.2. Customized evaluation and instruction from individual to individual

A diagram embeds a student's conceptual understanding, while evaluation by the expert module is trying to infer it. Thus evaluation questions need to be somehow mapped to domain-wide concepts. In order to track student's knowledge on each concept, it is necessary to register them in student model. A complete set of evaluation steps will be applied to the student's diagram at the beginning, as the domain-wide concepts in the student model is not determined. As the student finishes a problem, he/her student model will get updated, with some concepts being checked as passed. How to define a concept as mastered is not in the scope of this paper. The next time, the expert module should consult student's concept inventory before initializing the tutoring process. For example, we have implemented six questions in the expert module in the Thermo Cycle Tutor. However, for the student who has understood phase change temperature, how to use the reference form to locate the value, and how it should appear in a T-v diagram, expert instruction would skip question 4, which checks the label of phase change temperature in the future T-v diagram evaluation. Or, the tutoring system could give different feedback when error is found in the mastered concept.

5.3.3. Instruction needs to separate conceptual knowledge from procedure expertise

There are some practical issues when the evaluation engine assesses a student's drawing based on the elements defined in the expert module. How to handle these issues will affect the usefulness and quality of instructional feedback, student's engagement and their learning gains.

In most cases, when a student starts to frame a problem, he/she doesn't have a clear idea of what information needs to be drawn, and what might be a proper way to represent it. So a sloppy drawing with some incomplete elements might be submitted to the system for help. In order to provide the most useful instructional feedback, the tutor is desired to "read" information from the sloppy drawing. The information includes what might be the student's intention, what knowledge he/she might have known or not known and what other knowledge needs to be further determined from the drawing.

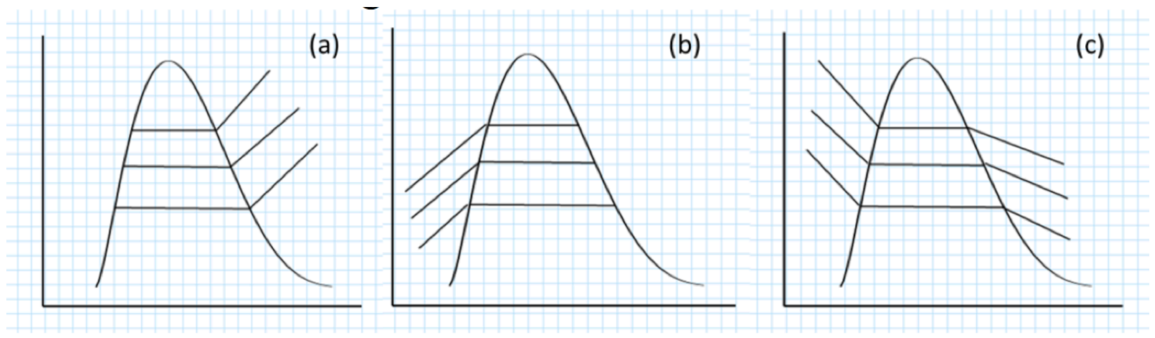


Figure 5.2. Examples of wrong drawings on refrigeration cycle T-v diagram. (a) and (b) Incomplete pressure line. (c). Wrong pressure line representation which uses a negative slope.

The student might not be able to represent his/her conceptual understanding in the drawing in a correct way at the beginning. However, when the student gets familiar with the procedure or gains some expertize on how to represent the knowledge, he/she can focus more on the conceptual part. So a tutoring system needs to set apart these two types of questions, and give instructional feedback separately. Figure 5.2 (a) and (b) show two examples of sloppy drawing where a vapor dome and three coupled lines were present. To be considered as a correct representation of a pressure line, three connected segments should be included. However, the incomplete drawing still implies that the author thought there were three pressure lines. Assume three pressure lines are the correct answer in expert solution. In this case, the tutor's instructional feedback should focus on how to help them to construct a pressure line, instead of correcting the number of pressures because zero "true" pressure lines were detected in the diagram.

Figure 5.2 (c) shows an incorrect representation of pressure line, since slope of the two sided-lines should be positive. For many beginners, they tend to borrow the shape that they learned in P-v diagram, which is negative, and apply it to T-v diagram. Even though the tutor cannot detect correct pressure lines, it should be able to probe students' intention, and give them appropriate instruction such as "This is not a P-v diagram. Would you like help on drawing pressure lines in a T-v diagram?"

In order to facilitate this strategy, we provide some guidelines when implementing tutor evaluation engine.

5.3.3.1. Diagrams require different tolerances at different levels of evaluation.

As we mentioned earlier, instruction could be based on evaluation of a three-level hierarchical structure of the drawing. A different tolerance could be assigned to each level. A larger tolerance could be used to detect if a basic structure is present. For instance, in order to check the number of pressures, the tolerance for gaps between line segments could be a larger value. When it passes the Level 1 check, and goes to the next level, which checks the representation correctness, this tolerance would decrease, and any big gap would need to be filled. Two examples with incomplete pressure lines (Figure 5.2 a and b) would pass the number of pressure check, given a higher tolerance on recognizing a pressure line. After a student passed the Level 1 check, the incomplete pressure line issue would be addressed due to a lower tolerance on how a pressure line should be drawn.

5.3.3.2. Diagrams' inherent ambiguity can be resolved with Q&A.

Due to the intrinsic complexity and ambiguity of a drawing, it is safer to confirm the information that is conveyed in a drawing with some text inputs. For example, if a drawing fails on a *number_of_pressures* check, a multiple choice question pops up and asks student to choose how many pressures in the system. If it is correct, it indicates that the student's conceptual understanding is correct, but some procedural issue causes the failure, e.g., he/she accidentally clicks the submit button without finishing the pressure line. Another example is shown in Figure 5.3. The student did a good job on drawing pressure lines, labeling pressure and phase change temperature, and anchoring points on pressure lines to show the state changes in each component. However, feedback from the tutor said, "There appears to be some misconceptions about the specific volume change in a compressor." Then the tutor directed the student to three multiple-choice questions regarding pressure, temperature and specific volume change in a compressor. The student answered all the questions correctly and was told to "modify state 3 and 4 to reflect this." These successful answers imply that the student understood the knowledge in a compressor, but didn't incorporate it into the drawing.

5.3.3.3.Conceptual and procedural performance in diagrams can be tightly coupled.

This problem is critical and stems from the fact that some aspects of constructing the drawing can make it difficult to edit elements later. This situation can frustrate a student if it occurs late in the problem solving. As is shown in Figure 5.3, after the student realized state 3 should have a larger volume than state 4 (which means state 3 should appear on right side of state 4 in the T-v diagram), it is impossible for the student to change it in the diagram because there is no room. However, the student would not realize this issue until he/she reached this step with no prior experience on this type of problem. To alleviate this form of unnecessary frustration, when a particular problem is initialized by student, the evaluation engine should be able to load some practical expertise information about the base objects, e.g., the shape of the vapor dome should not be too thin and the distance between the horizontal lines should be greater than some threshold.

5.3.3.4.New error can be generated as a by-product after fixing an observed error.

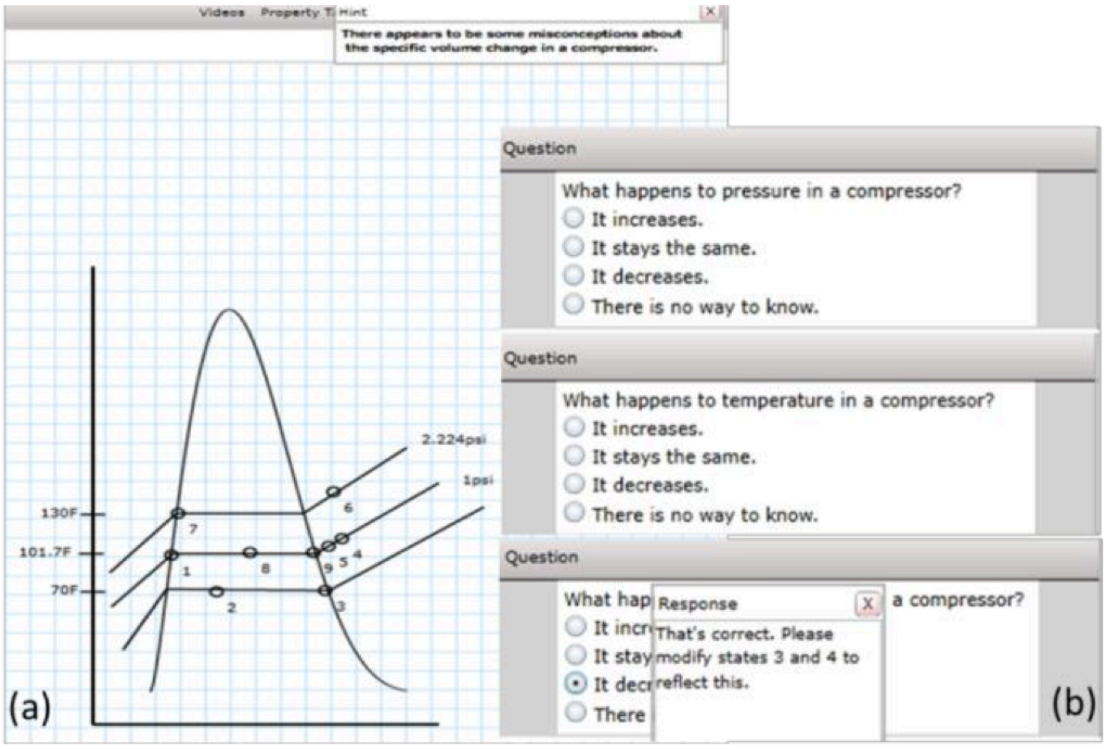
For most cases, spatial objects are correlated. The relative position of one object to another may be affected if we update its relation to a third object. It happens a lot after student fixes one error based on tutor's instruction, another error would come up because of the new operation. This tells us throwing all errors at once is not a good idea, as new errors would be generated as student proceeds. Instead, correct their misconception one at a time by some text-based input, and the knowledge should allow them to fix at least one error on the drawing each time. However, if new evidence is found that an error occurs on a concept that has been thought as mastered, how should tutor respond? There is a big chance that this error is generated as the by-product of fixing another error, where student is unaware of this new one. In this case, tutor's instruction should not focus on conceptual correction, but serve as a reminder, e.g., "Please check the expansion valve." Also, the student model should have a different index to store this type of error for future coaching and post analysis.

5.4. Conclusion and Future Work

In this chapter, we discussed some cross-domain pedagogical strategies in Diagram-based ITSs, which we learned from two systems we developed: Statics Tutor and Thermo Cycle Tutor. Even though they bear different underlying evaluation structures, it is noteworthy, due to their diagrammatic representations, they share similar features in pedagogy design.

In particular, instructional feedback needs to be mapped to a hierarchical understanding of the diagram. Personalized evaluation is desired which is based on student's current knowledge state. Also, it should be able to separate conceptual knowledge from procedure expertise. In order to achieve that, we proposed some suggestions, such as 1) allow different tolerances at different levels of evaluations, 2) use Q&A to reduce ambiguity, 3) determine if conceptual knowledge can be applied by procedure expertise in current drawing, and 4) determine if the new evidence violates the existing understanding of the student. Though these suggestions are mainly abstracted from quantitative diagrams or diagrams contain point-wise relation, we believe they are still useful for other types of diagrams that don't consider about the spatial location of its components. For instance, for diagrams containing geometric shapes and links among them (Block Diagram), we can still customize diagram evaluation based on student's current understanding. E.g., if a student has already mastered the concept of inheritance relation in Entity-Relationship diagram, tutoring system might ignore errors from this type of connection in the future problem. Also, the idea of separating conceptual knowledge from procedure expertise is applicable to any type of diagram. For instance, different types of shapes are used to model different elements in process flow diagram, such as a diamond as a decision node. Thus, students need to get used to those shapes in order to convey their knowledge smoothly.

We propose the future authoring tool for Diagram-based ITSs will support the above-mentioned pedagogical strategies. The tool will allow instructors to define a) concepts in the knowledge base, b) objects and tolerances in each hierarchical level, c) evaluation pieces which link to one or more concepts and d) guidelines of procedural expertise. In the next chapter, we will discuss another aspect of creating authoring tool: how to represent diagram's knowledge from a general perspective, and evaluation process to diagnose student's diagram. Also, we will give a more practical way to include pedagogical instructions to handle diagnostic output. We use different schemes to categorize pedagogical instructions in chapter 6, since Block Diagrams have different features from quantitative diagrams.



There appears to be some misconceptions about the specific volume change in a compressor.

What happens to pressure in a compressor?
- It increases.
- It stays the same.
- It decreases.
- There is no way to know.

What happens to temperature in a compressor?
- It increases.
- It stays the same.
- It decreases.
- There is no way to know.

What hap… a compressor?
- It incr…
- It stay…
- [✓ It decr…]
- There…

That's correct. Please modify states 3 and 4 to reflect this.

Figure 5.3. (a). A refrigeration cycle T-v diagram. (b). Three windows that displayed questions about pressure, temperature and specific volume change in a compressor.

REFERENCES

Conati, C., Gertner, A., VanLehn, K., & Druzdzel, M. (1997). On-line student modeling for coached problem solving using Bayesian networks. In A. Jameson, C. Paris, & C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference* (pp. 231–242). Vienna: Springer.

Futrelle, R. P. (1990). Strategies for Diagram Understanding: Generalized Equivalence, Object/Spatial Data Pyramids, and Animate Vision. In *10th ICPR International Conference on Pattern Recognition* (pp. 403–408).

Guo, E., Gilbert, S., Jackman, J., Starns, G., Hagge, M., Faidley, L., & Amin-Naseri, M. (2014). StaticsTutor : Free Body Diagram Tutor for Problem Framing. In *Proceedings of the 12th International Conference of Intelligent Tutoring Systems.* (Vol. 8474, pp. 448–455). Honolulu, HI.

Guo, E., Jackman, J., Gilbert, S., Hagge, M., Starns, G., Faidley, L., & Amin-Naseri, M. (2015). *Decision-centric problem framing tutor for thermodynamics (working paper)*.

Hagge, M., Amin-Naseri, M., Jackman, J., Guo, E., Gilbert, S., Starns, G., Faidley,L. (submitted to *Advances in Engineering Education*). Decision-based Learning for Thermodynamics Phase Diagram.

Koffman, E. B., & Friedman, F. L. (1976). A computer-aided flow diagram teaching system. *SIGCSE Bull.*, *8*(1), 350–354.

Py, D., Alonso, M., Auxepaules, L., & Lemeunier, T. (2008). Design of Pedagogical Feedbacks in a Learning Environment for Object-Oriented Modeling. In *Proceedings of Educators Symposium at the 11th IEEE International Conference MODELS* (pp. 39–50).

Shah, P. (1997). A Model of the Cognitive and Perceptual Processes in Graphical Display Comprehension. In M. Anderson (Ed.), *Reasoning with diagrammatic representations* (pp. 94–101). Menlo Park, CA: AAAI Press.

CHAPTER 6

AN ONTOLOGICAL APPROACH TO SUPPORT KNOWLEDGE REPRESENTATION IN DIAGRAM-BASED INTELLIGENT TUTORING SYSTEMS

This chapter will be submitted to the journal *Computers and Education*
with author list:

Enruo Guo, Stephen Gilbert, Les Miller

Abstract

In this work, we use an ontological approach to represent both domain knowledge and an evaluation process for Diagram-based Intelligent Tutoring Systems (ITSs). These two forms of knowledge are quite different—the structure and content of a diagram itself vs. the process that an expert uses to analyze a diagram. Our approach is designed to ease the creation of Diagram-based ITSs by allowing knowledge re-use. Specifically, a general property ontology is provided which categorizes Block Diagrams into five subtypes, where each subtype is given properties based on its critical features. Knowledge representation of Block Diagrams is modeled by abstract classes, such as *BLOCK, REPRESENTATION, CONTENT, OPERATION* and *CONNECTION*, which provide a modular way to allow an evaluation system to parse the diagram and build an internal knowledge representation. To address the second overall form of knowledge, the evaluation process, we describe a diagram evaluation ontology which uses the class representation and includes three steps: 1) general property check, which evaluates if the given diagram satisfies the properties defined by the general property ontology; 2) block property check, which detects if each block is appropriately represented and includes the

permitted number of connections and connecting types; and 3) problem-specific check, which checks whether a student's diagram has similar information as the expert's solution in a given problem. Conceptual knowledge can be also evaluated in terms of conceptual connections between blocks. Furthermore, we extend this ontology by adding the pedagogical instructions to handle the outputs of the evaluation process. Applications in four domains are discussed (process flow diagrams, ER diagrams, circuit diagrams, and concept maps) to demonstrate that our approach is feasible in different areas. Finally, we give three case studies to validate our class representation and evaluation/pedagogy ontologies.

6.1. Introduction

There are barriers in developing an Intelligent Tutoring System (ITS). Authoring an ITS requires excessive efforts in modeling domain knowledge, either through rule-based or constraint-based approach, along with designing diagnostic steps, and developing pedagogical feedback to students (Murray, 2003). It is helpful to provide a standard structure for experts, so that they can build each piece modularly. Also, knowledge re-use is highly desirable, since it will eliminate redundant work and save time, if commonalities exist between the new domain and a previously modeled one. Therefore, an underlying support to an authoring tool that allows knowledge re-use and modularity becomes an good solution for experts to create an ITS with the least effort.

Diagnostic steps in ITSs are usually a part of the expert module, which specify the sequence in evaluation and components needing to be addressed by the tutor. Also, pedagogical activities are offered to resolve any errors in the problem-solving process. In general, diagnostic steps are domain-dependent and require expertise to design. For

Diagram-based ITSs, designing those steps becomes even harder. Unlike text-based inputs, diagrams can be drawn in different sequences of edits, though they should be perceived as an entire state by a tutoring system. However, some pieces are more fundamental, on which other pieces are based. At times, the sequence of adding elements does matter. Thus, evaluating a diagram becomes more challenging. Diagram diagnosis needs to incorporate both instructional expertise, such as domain knowledge, and pedagogical expertise about the diagram representation itself. Therefore, our research questions follow.

> ***Research Questions:*** Is there a general way to represent a diagram's domain knowledge that is applicable to various types of diagrams, to support the authoring of a Diagram-based ITS? Can we also create evaluation procedures and tutoring instructions on top of it?

We use class as an explicit form of knowledge representation for domain modeling in Diagram-based ITSs. Evaluation / pedagogy ontologies to articulate evaluation steps and expert's feedback are also provided. To demonstrate the validity of the ontologies, we will validate the process involved in creating a tutor design from scratch using the class representations and ontologies. Then two examples of generalization will be discussed: 1) a big transfer: the process of creating the three-step evaluation ontology in a new area based on the existing ontology and the changes that are needed, and 2) a small transfer: the process needed to create the evaluation ontology for a new problem within the same domain. Thus, by introducing modular structure and enabling the knowledge re-use, we provide an important methodology that can be used to design future Diagram-based ITSs.

6.2. Literature Review

6.2.1. Introduction to ontologies

In Artificial Intelligence, knowledge representation is the method for encoding knowledge in an intelligent system's knowledge base. As one of many knowledge representation methods, ontologies have been used in many fields in the last decades. In order to facilitate knowledge sharing, ontologies are used to organize related concepts under a common specification. According to two widely adopted definitions provided by Tom Gruber (1993) and Willem Borst (1997), Jakus et al. (2013, p 29) defined ontology as following:

"A formal and explicit specification of a shared conceptualization."

The purpose of creating ontologies includes: 1) knowledge sharing; 2) domain knowledge re-use; 3) making domain assumptions explicit; 4) separating domain knowledge from operational knowledge; and 5) analyzing domain knowledge (Noy & McGuinness, 2001).

An ontology has a few key aspects. It includes conceptualizations by using abstract models to describe the real world. It defines explicit concepts and relations. It uses formal language that is intended to be readable by the computers. And lastly, it is shared by a group of users who commit to a set of terms and consensual knowledge (Studer, Benjamins, & Fensel, 1998).

According to previous research (Gomez-Perez & Corcho, 2002; Khoo & Na, 2006; Studer et al., 1998), the components of ontologies generally include: a) concepts, classes, collections, set or types; b) objects, individuals, instances or entities; c) attributes, properties, or features of concepts; d) attribute values; and e) relations among classes or/and objects. An ontology is typically built on top of a taxonomy, and then expands to a richer network by adding semantic relations and additional components such as functions, restrictions or constraints.

As Noy & McGuinness (2001) have mentioned, there are some fundamental rules in creating ontologies:

1). There are always alternatives when modeling a domain. Criteria including potential applications, extensibility and maintainability could apply in choosing the appropriate approach.

2). Ontology development needs an iterative process. By applying it to real-world problems or by discussing it with the domain experts, we can evaluate and revise our initial design, which continues through the entire lifecycle.

3). The abstract concepts need to maintain real-world objects' relationships in the domain of interest.

Four main stages are involved in the ontology lifecycle: 1) Specification, where the purpose and scope is defined; 2) Formalization, where models are built according to the requirements from specification stage; 3) Maintenance, which tracks changes and evolution, and detects any inconsistency as well; and 4) Evaluation, which checks if the existing ontology meets the requirements (Guarino & Welty, 2002).

6.2.2. Ontology as a way of domain modeling in ITSs

One of the biggest obstacles in ITSs development is the cost of creating domain model from scratch. One solution is domain ontology engineering (Zouaq & Nkambou, 2010). Basically, two main advantages are involved: 1) knowledge sharing and re-use between any ontology-friendly environments, and 2) enabling knowledge extraction automatically, which is called "Ontology Learning" (Zouaq & Nkambou, 2010). It is believed ITS could benefit from an integration with the Semantic Web by better reuse of its educational components and sharing (Zouaq & Nkambou, 2010). *Educational Semantic Web* (Aroyo & Dicheva, 2004) is a good example. It uses ontologies to index and structure the learning content.

6.2.3. Task ontology

Effort has been made by other researchers to improve the authoring process by applying ontologies in some ITSs (Ikeda et al., 1997; Chen et al., 1998). For example, a task ontology that captures semantic features in the system can be used to represent problem solving descriptions. According to Ikeda, Seta, & Mizoguchi (1997), the advantages of integrating a task ontology with authoring tools include: 1) Descriptiveness and readability; 2) conceptual level operationality – it simulates the problem solving at the conceptual level; and 3) symbol level operationality – it can make the task description runnable by converting it to symbol level code.

Ikeda et al. (1997) discussed an ontology-based authoring tool. The tool allows the user to design the skeleton of the training process, add training material, and adjust the process control in details. Three layers of ontology are defined: a) core task ontology with the least dependency lays foundation for other layers; b) task-type specific ontology, on the second level, is a theory of concepts/vocabulary to describe task models; c) task-domain ontology, on the top level, describes domain models from the task-type perspective. Thus, an ontology that contains both domain knowledge and task-specific information can be a good solution to capture both a diagram's representation and an evaluation process for Diagram-based ITSs authoring tools.

6.3. Our Approach

6.3.1. Overview

In this section, we will have our focus on one type of diagram, which contains geometric shapes and connections to demonstrate relationships, flow, or pattern. As each geometric shape can be viewed as a block, we call it a Block Diagram. Many diagrams belong to this category, such as process flow diagrams, UML diagrams, ER diagrams, concept maps and circuit diagrams. Even though, in some diagrams, the geometric shape represents conceptual meaning under a given domain, whereas some don't, they share similar structures with respect to the steps of evaluation, e.g., "Are all critical blocks included?", "Are connections among blocks appropriate?" and so on. In our work, we try to make the creation of Diagram-based ITSs, based on Block Diagrams, easier and more modular to manipulate.

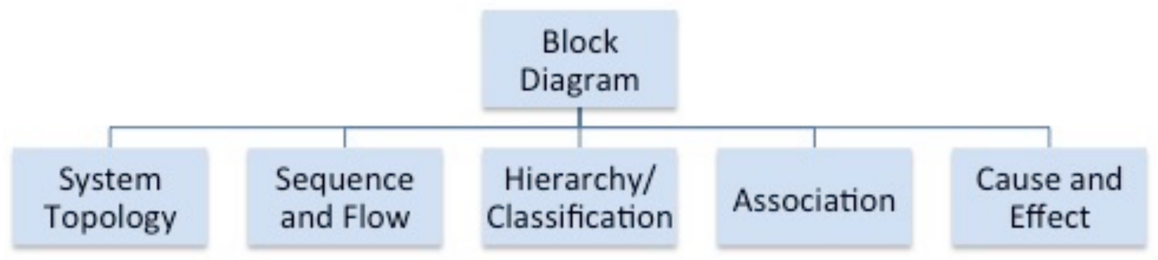


Figure 6.1. A classification of Block Diagrams, adapted from Nakatsu (2010).

<u>*6.3.1.1 Overview of general property*</u>

First of all, five subtypes of Block Diagram were defined based on the work of Nakatsu (2010), shown in Figure 6.1. Each subcategory has some intrinsic features that can be conveyed through diagraming. E.g., one subcategory in Block Diagram is *Sequence and Flow*, which indicates arrows are needed to show the flow from one object to another. If the given diagram uses lines instead of arrows in connecting objects, it would fail to be a *Sequence and Flow* subtype. Therefore, some general evaluation needs to occur to detect the validity of the subtype, before going through diagnosis of problem-specific knowledge. Example diagrams in each subtype and their properties are described below. Such information is critical when creating a Diagram-based ITS in a new domain. Thus, we use "general property" to refer to intrinsic properties that carry some conceptual meanings or restrictions based on their subcategory type.

6.3.1.2 Overview of block property

Also, some diagrams use specific representations to convey the meaning of objects. For instance, in an entity relationship (ER) diagram, a rectangle is used to represent an entity, an oval is used for an attribute and a diamond for a relation. A person familiar with these diagrams can immediately transfer the shapes to their conceptual meanings. In addition to that, each representation has certain “rules” to capture its domain knowledge for an object, e.g., to which pieces it must be connected, to which pieces it could be connected, and the minimum or maximum number of connections it can have. This information is dependent on the particular type of Block Diagram. We call this sort of information the “block property”.

6.3.1.3 Overview of problem-specific property

Finally, we build the diagram representation and compare it with the expert’s solution. There are two major questions: What kind of information in the diagram is critical in terms of representing the student’s understanding? How can we make it modular so that it works for different subtypes in Block Diagrams? After evaluating many types of Block Diagrams, we have identified several kinds of information: number of blocks and number of connections in the diagram, and information on each block and each connection. Some of the blocks are more critical, i.e., they convey important knowledge, which cannot be replaced or neglected. For example, an entity node in ER diagram is critical, which generally needs to be implemented as a database table. However, an attribute node is not as critical as an entity node, as it represents the property of an entity that is only a column within a database table. Different attribute nodes can carry similar information. E.g., in an online shopping user profile within ER modeling, an entity called *user* can have either an attribute *date-of-birth* or *age*, which both capture how old the user is. Also, in some domains, connections can be associated with some conceptual meanings. For example, in a process flow diagram, connections pointing to/out a decision node can be interpreted as a decision-making condition. Any connection errors involved in this node would indicate a misconception in decision-making process. In summary, we call the information mentioned above "problem-specific".

Here, we propose an evaluation ontology to deal with the five subcategories in Block Diagram (See Figure 6.3 and detailed discussion in Section 6.3.3). The ontology contains three branches, which correspond to the three types of information we discussed above. They are:

1) ***general property check***, which checks if the diagram satisfies the properties of the subcategory it belongs to;
2) ***block property check***, which checks if each block meets the rules based on its domain knowledge; and
3) ***problem-specific check***, which checks if the information in terms of instances of blocks and connections is correct, according to expert's solution.

Then next, we apply this ontology to three domains to demonstrate its feasibility. Furthermore, we give three case studies to show the ease of creating evaluation procedure for Diagram-based ITSs. More importantly, we want to show that this method can be implemented as the default knowledge in the future authoring systems for Diagram-based ITSs.

6.3.1. Definitions

To make our discussion clear, we will give some definitions.

An **Information graphic** or **Infographic** is a graphical visual representation of information, data or knowledge intended to present information quickly and clearly (Smiciklas, 2012). Nakatsu (2010) suggested information graphics have a bigger scope than diagrams. Based on his explanation, diagram is an information graphic that consists of geometric shapes and connections such as lines or arrows. The purpose of diagram is to show entities, such as things, people or activities, and their interrelationships in a qualitative way. However, his definition on diagram excludes some important information graphics in STEM such as quantitative graph. Therefore, the diagram defined by Nakatsu (2010) is called Block Diagram in our definition.

A **Block Diagram** is an information graphic that is made up primarily of geometric shapes with connections by lines or arrows. Some texts might be attached along the connections to show dependency relation, operation and logic condition, etc. The spatial location of each element in the diagram is not considered, whereas only the appearance of the object and its connectivity to other objects are of interest. For our purposes, we assume that all diagrams of interest are members of established or widely-used diagram types, e.g., those used in STEM fields or business. We adapt the classification of diagrams from Nakatsu (2010) and choose five categories as is shown in Figure 6.1.

6.3.2. Classes in a Block Diagram

We define classes that will be useful in capturing information in Block Diagram, i.e., a Block Diagram can be represented as a few instances of several classes. Two important classes are *BLOCK* and *CONNECTION*. The concept of block and connection as definition components of a Block Diagrams is not new. The Systems Modeling Language (SysML) is a general-purpose modeling language extended from Unified Modeling Language (System Modeling Language, 2015). Elements in SysML include Block and Connector. Block is the modular unit that describes the structure of a system, which includes both structural and behavioral features, whereas Connector can be used to define relationships between Blocks. Detailed information about the link can be associated with the connector properties that contain complicated visual representations. Also, in SysML, specific keywords are defined to capture different property types or operations. For instance, a "part" is a property defining the local usage of a block. A block that represents the definition of a wheel can be used in different ways. A front wheel and a rear wheel can have different usage. For each individual usage, content-specific values or constraints can be defined, e.g., 25 psi for the front tires and 30 psi for the rear tires. However, our focus is not to design sophisticated classes that model system properties or behaviors. Instead, a simple but clean class representation is desired to capture general properties that a Block Diagram might possess.

Based on the definition of Block Diagram, we perceive it as a set of blocks and connections. Therefore, classes that capture the features of blocks and connections are developed.

a) Class BLOCK and its related classes.

A block is a basic element in a Block Diagram. The geometric shape of a block cannot be either subdivided, or partially represent the object. Within a block, there shall be no other connections. To model the class *BLOCK*, three attributes are necessary: *REPRESENTATION, CONTENT* and *OPERATION*. Note each of them is an individual class, which we will discuss in the following.

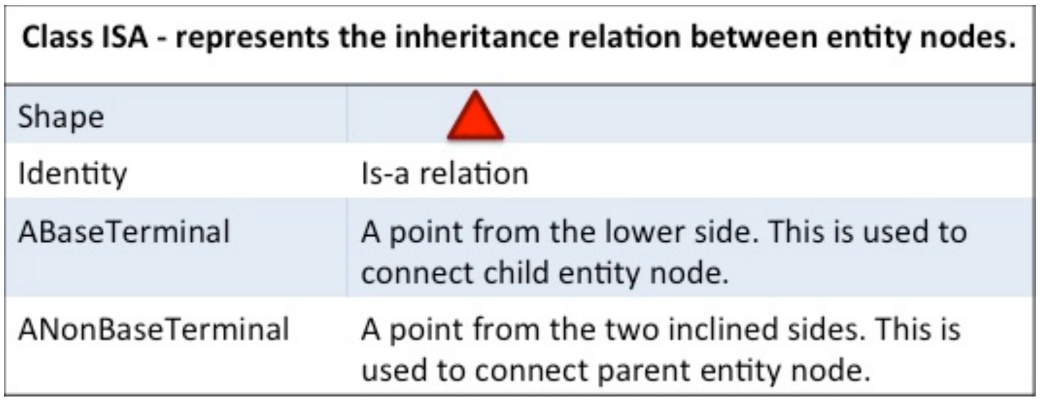
REPRESENTATION: A geometric object that represents the block. A shape is associated with a representation, which is domain-dependent. For instance, a rectangle represents a process step in process flow diagram, whereas it stands for an entity in entity-relationship diagram. In some domains, representation can reveal the identity of the block. Furthermore, considering how it can be connected to other block, there can be strict requirements on the location of the terminals, and how many numbers of connections can be made. Each domain needs to implement a set of representations to facilitate diagramming the knowledge, as well as making the knowledge re-usable. Here, we list ten properties that a *REPRESENTATION* class would posses.

- Identity: the identity of a *REPRESENTATION*, if the shape can reveal it. It is a string variable type. For instance, the identity of a rectangle in a circuit diagram is a resistor.
- Shape: a geometric object available from the drawing palette. It is the geometric construction of a *REPRESENTATION*. Internally, it is an object with a name such

as rectangle, a function that is used to generate it, and a few important parameters that decide its size and location.

- Num_of_Terminals: the maximum number of connections that a *REPRESENTATION* can have. The value type of this property is either ND (not defined) or an integer.
- Terminals: the contact points of a *REPRESENTATION*, based on user-defined terminal name. The value type is either ND or a list of string.
- Loc_of_Terminals: the location of the contact points of a *REPRESENTATION*. It pairs the user-defined terminal name with its spatial location. The value type is either ND or a Hashtable with a string type as the key and a *POINT* type as the value.
- Equiv_Terminals: terminals with similar properties that serve as alternatives for members within the same group. The value type is either ND or a list of terminals.
- ARandomTerminal: any point on the perimeter of the shape. This property is used to capture a contacting point from a random position. The value type is a *POINT*.
- AVertexTerminal: any point at the vertices of the shape. This property is used to capture a contacting point from the vertices position. The value type is a *POINT*.
- ABaseTerminal: any point at the lowest side of the shape. This property is used to capture a contacting point from the base position, which usually applies to shapes with a horizontal side at the bottom, such as a triangle. The value type is a *POINT*.
- ANonBaseTerminal: any point not at the base side of the shape. It is a complement of ABaseTerminal. The value type is a *POINT*.

Note: not all the properties are available in each domain. An example representation class *ISA* is shown in Figure 6.2, which defines four properties. For those domains that have limited terminals, Num_of_Terminals, Terminals, Loc_of_Terminals and Equiv_Terminals must be defined together. For instance, a resistor in circuit diagram has two terminals in the long axis direction.



Class ISA - represents the inheritance relation between entity nodes.	
Shape	▲
Identity	Is-a relation
ABaseTerminal	A point from the lower side. This is used to connect child entity node.
ANonBaseTerminal	A point from the two inclined sides. This is used to connect parent entity node.

Figure 6.2. A representation class *ISA* in ER diagrams.

CONTENT: The texts within a *BLOCK*. Based on different applications, the texts may contain a variety of information, such as conceptual names, labels, functions, values or properties associated with the block. We listed four types of information below, where *CONTENT* may involve in one or up to four kinds in real applications.

- Label: the texts that reveal the name of a *BLOCK*. E.g., a label with "personnel" can be put inside of an entity block in an entity-relationship diagram. It is a string variable type.
- Value: the numerical value that relates to a *BLOCK*. E.g., 100 V can be attached to a voltage node in circuit diagram. It is a string type.
- Function: the function that belongs to a *BLOCK*. E.g., functions are specified in each class of UML diagram. It is a string type.

- Property: the property or attribute that is associated with a *BLOCK*. E.g., property is specified in each class of UML diagram. It is a string type.

OPERATION: An action based on any arrow/line pointing out from the *BLOCK*. This property mainly applies to sequence/flow or semantic type diagrams, whereas it is rare for the relation/hierarchy types diagrams. It contains three components:

- Label: the texts that reveal the action. For example, a label with "to the UV clean system" is attached to a water pump in a process flow diagram. It is a string variable type.
- Terminal: the contact point where the operation points out from a *BLOCK*. It is either a ND or a *POINT* type.
- Operation type: the shape of the operation. It is either a *LINE* or an *ARROW*.

<u>*b). Class CONNECTION and its related classes.*</u>

CONNECTION captures how two *BLOCKs* are linked. Four components are considered: the two connected nodes, the linkage type between them and labels associated with the link.

- Node_1: the first *BLOCK* associated with the connection. It is a *BLOCK* type.
- Node_2: the second *BLOCK* associated with the connection. It is a *BLOCK* type. Node_1 and Node_2 are exchangeable.
- Label: the texts along with the *CONNECTION*. It is a string variable type.
- Connecting type: the shape of the *CONNECTION*. It is either a *LINE* or an *ARROW*.

LINE: a straight connection between two points.

- Point_1: the first end point of a *LINE*. It is a *POINT* type.

- Point_2: the second end point of a *LINE*. It is a *POINT* type. Point_1 and Point_2 are exchangeable.

ARROW: a line with an arrowhead pointing to one end.

- StartPoint: the start point of an *ARROW*. It is a *POINT* type.
- EndPoint: the end point of an *ARROW*. It is a *POINT* type.

POINT: an exact position on a plane surface.

- X_location: the x coordinate of a *POINT*. It is a float type.
- Y_location: the y coordinate of a *POINT*. It is a float type.

6.3.3. Block Diagram evaluation

In our work, we design an evaluation process that contains three steps (see yellow nodes in Figure 6.3). The first step is to check general properties, which is problem-independent and domain-independent. At this stage, an expert solution is not necessarily needed. The general property check evaluates whether the given diagram satisfies the properties of a subcategory within a Block Diagram, e.g., "Is this diagram a valid ER diagram?" Second, the system runs a block property check. It is problem-independent too. However, it relies on domain knowledge, which checks the properties associated with each block's representation. Such property includes number of connections and legal connection type: "Is this diagram well-formed according to the rules of this domain?" Third, it does problem-specific check. In this step the system compares an expert solution with the student's drawing by retrieving *block-wise* and *connection-wise* information, where the classes we defined earlier will be used to construct knowledge representation.

Note the evaluation engine goes through each step sequentially. If a student's drawing fails on general check, the tutor will stop and return pedagogical instructions to correct the error. If all items in general check are satisfied, tutor will continue to the next step. Below we will discuss each check in detail.

6.3.3.1 General property check

Figure 6.4 shows a general property check ontology for Block Diagrams. We explain it below. This ontology is based on diagram categorization research by Nakatsu (2010), which seeks to help tutor authors answer the question, "Is this a valid Block Diagram? Does it meet additional requirements from the subcategory it belongs to?" For example, what is required to make a valid inheritance hierarchy diagram? This general property ontology is used to validate whether a give diagram has a correct diagramming format with an appropriate connectivity links. We will discuss this ontology from three perspectives.

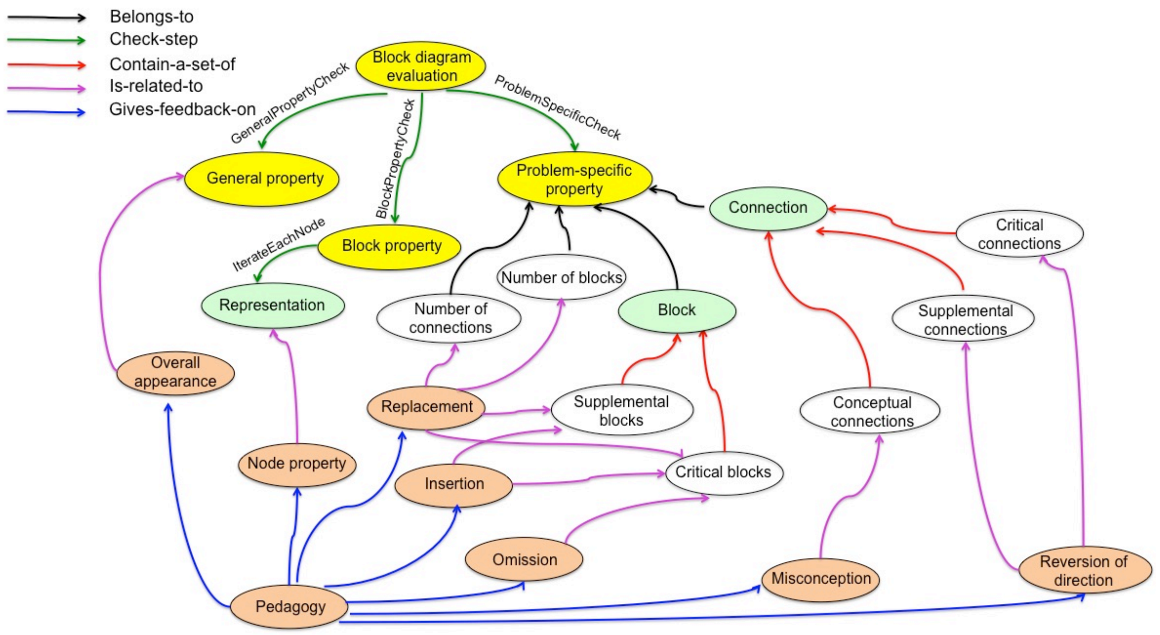


Figure 6.3. An ontology that includes evaluation steps and pedagogy types to handle evaluation process outputs in Block Diagrams. Yellow nodes: 3-step evaluation process. Orange nodes: pedagogical instruction types. Green nodes: abstract classes that need to be implemented based on domain-dependent knowledge. White nodes: data structures that include information from a diagram. Yellow, white, and orange nodes apply across domains; green nodes are domain-dependent.

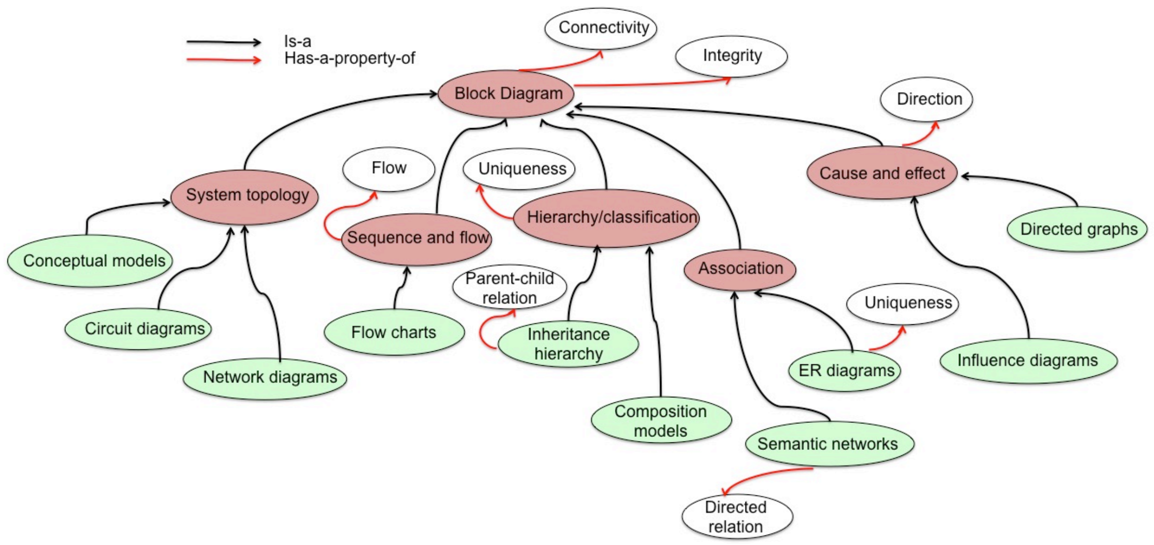


Figure 6.4. An ontology that includes general properties in Block Diagrams. Red nodes: top level subcategories in Block Diagrams. Green nodes: second level subtypes under each subcategories. White nodes: properties associated with Block Diagrams and their subcategories.

1) Five subcategories at first level

The five subcategories of Block Diagrams were modeled by the "is-a" hierarchy among the dark-red ovals, which were adapted from Nakatsu's (2010) work. This first level includes:

a. System topology, which uses a conceptual model to represent the organization of components. Examples include network diagrams, circuit diagrams, and conceptual models. In this type, only lines are required to connect to objects.

b. Sequence and flow, which shows temporal or chronological orders. Examples include various flowcharts. In this type, directed lines are used to indicate the sequence or flow from one object to another.

c. Hierarchy/classification, which shows a taxonomy, organization, or composition of a system, by using a downward direction to indicate the parent-child relation. Examples include composition models, hierarchy, etc. In this type, directed lines are used to model the parent-child relation in an inheritance hierarchy diagram.

d. Association, which shows how objects are related to one another. Examples include semantic networks and ER diagrams. As discussed above, in a semantic network, arrow is used to model the directed relation between nodes. In ER diagrams, entity nodes are not allowed to have duplications, i.e., every rectangle in ER diagrams must have unique content.

e. Cause and effect, which depicts causal relationships between objects. In this type, arrows are used to show direction of the cause and effect relation. Examples include directed graphs and influence diagrams.

2) Second level "is-a" relation

The second level "is-a" relation (from green ovals to red ovals) was created based on the definition of each subcategory and some examples from Nakatsu's (2010) work. This level gives concrete examples which can be directly referred to when creating a new ITS. For instance, this ontology shows that an ER diagram belongs to the *association* subtype. If an expert wants to create an ER diagram tutor, properties associated with an ER diagram can be retrieved from this ontology and then be evaluated during the general property check step.

3) Properties of Block Diagrams and their subcategories

Our contribution to this ontology consists of summarizing the major properties of each subtype, i.e., what basic features must be included, and what checks must be made by the tutoring engine (see white ovals in Figure 6.4). Based on our observation, a Block Diagram should satisfy two properties:

a. Connectivity, which checks if there is any outlying block that has no connection to other blocks. Also, it checks whether there is an isolated group of blocks that cannot be reached by other blocks. This property indicates there should be at least 2 blocks in a Block Diagram.

b. Integrity, which checks whether there is a block that is self-connected, i.e., two terminals from one block are connected directly. In general, self-connection is not common in many diagrams.

In addition, based on the definition and properties of each subtype, there are a few other checks for some of them. In the *sequence and flow* subtype, a "flow" check is required, which detects whether arrows are used between objects. In the *cause and effect* subtype, a "direction" check is needed, which examines whether objects are linked by directed line/arrows to reveal the casual and effect relation. In *semantic networks*, a "directed relation" should be used to model relations between nodes. In *inheritance hierarchy*, a "parent-child" relation is designated by an arrow pointing from a child node to a parent node. In the ontology, different phrases are used in different subcategories to describe the same geometric shape (in this case, an arrow). This is because we think using conceptual meaning is more reliable and straightforward than simple pictorial representation when constructing the ontology. In *hierarchy/inheritance* subtype, a "uniqueness" check is applied to find whether there is any duplicate block with same content, as it is rare to have same concept/component appear more than once. In *ER diagrams*, the "uniqueness" check is used to detect if any entity node has been reproduced in the same diagram, as duplicated entities are not allowed in ER diagram. Note that blocks are treated as identical if they share same content and representation.

6.3.3.2 Block property check

Having described the general property check, which is domain-independent and problem-independent, we move now to the block property check, which is still problem-independent but does depend on the domain knowledge of how a block should be correctly represented and how it should be validly connected. In particular, it checks the properties associated with each block's representation, such as the allowable number of connections and permitted connection type. The evaluation engine will iterate through each block in a student's diagram to do this type of check. Since it is domain-dependent, we use an example to illustrate how it works (Figure 6.4). The detailed block property check within three subcategories will be discussed in Section 6.4.

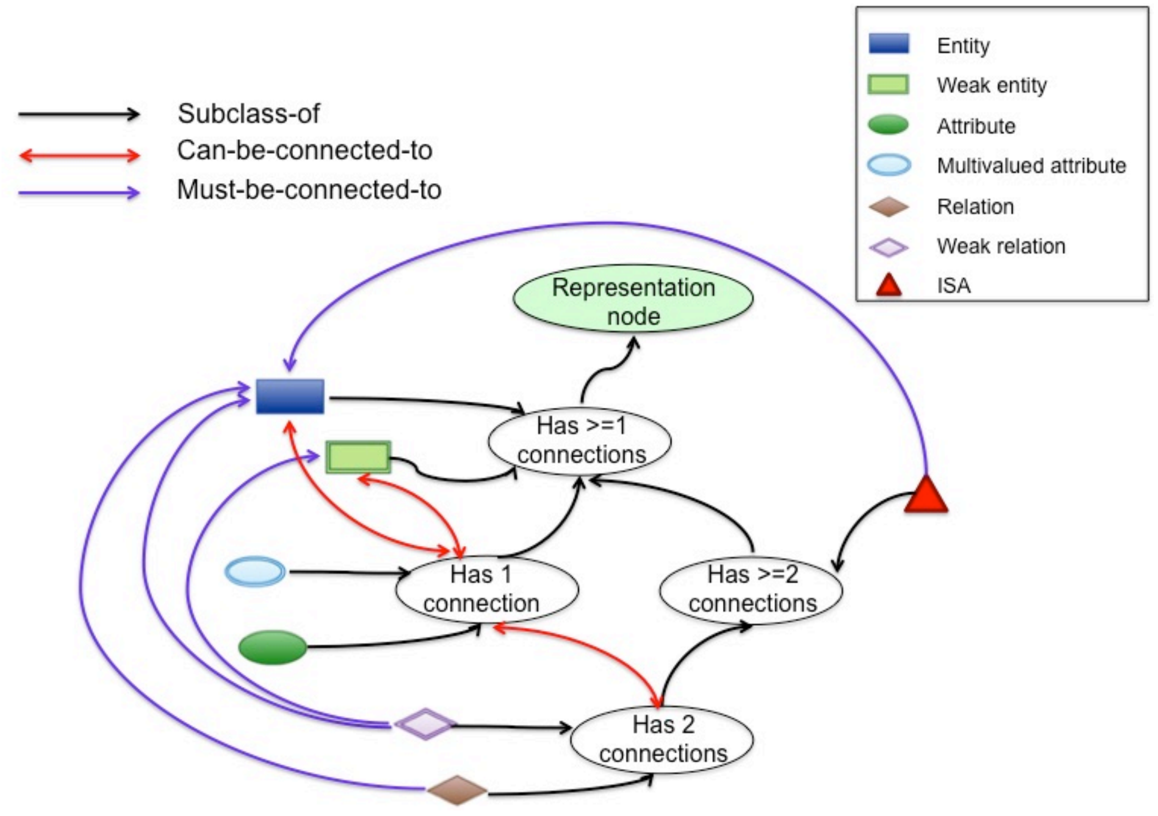


Figure 6.4. Block property ontology in ER diagrams.

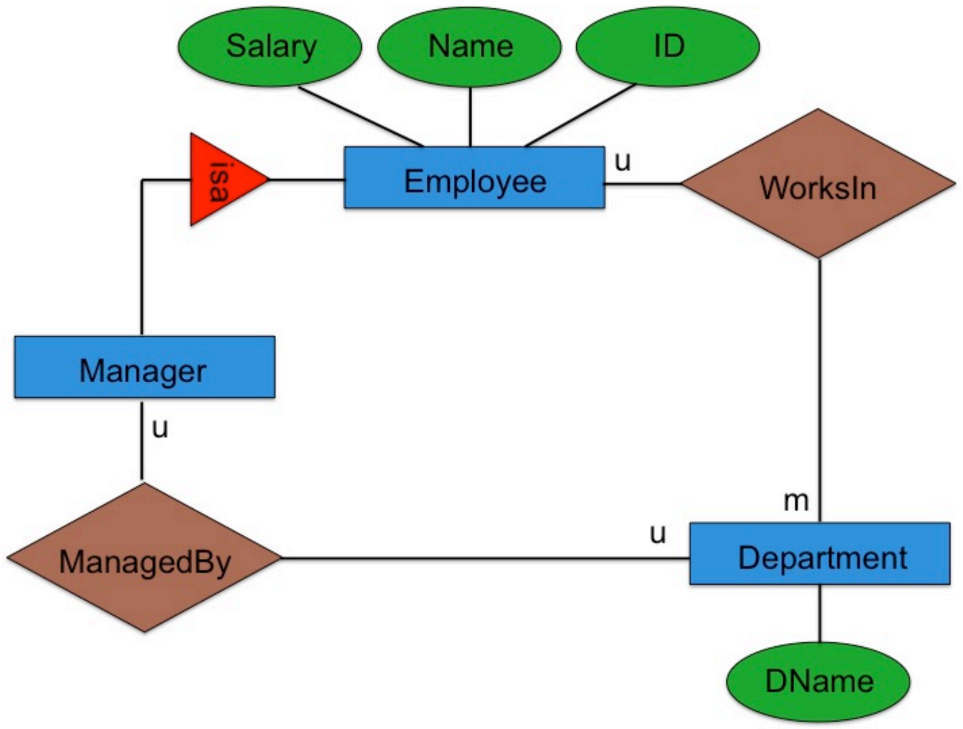


Figure 6.5. A personnel database ER model.

After the diagram meets the general properties we defined in Figure 6.3, it will be forwarded to the block property check in the second branch. The Representation class (green oval under block property) will be iterated one at a time. The connecting rules for each representation class is dependent on the domain knowledge. Thus, a separate block property ontology is required to handle that check. When the evaluation process reaches the block property check, it will move to the domain-dependent block property ontology to validate each representation object. Thus, Figure 6.3 represents the domain independent ontology, with the domain dependent components being represented in further sub-ontologies such as Figure 6.5. Here, we give an example: a block property ontology in an Entity-Relationship (ER) diagram. As is shown in Figure 6.4, three types of relationship are defined. This ontology is an example we created based on our understanding in ER modeling. It may not contain a complete relationship among blocks.

Here, we only use it as an example to demonstrate functions in block property check. The black arrows specify the parent-child relations between nodes. Four white nodes, which have information about the number of connections, organize the block representations in ER diagram into four types: has_one_connection, has_one_or_greater_than_ one_connections, has_two_connections, has_two_or_greater_ than_two_connections. The red double-ended arrows specify the connection type that a representation can be connected to. This is used to capture the knowledge that an attribute or multivalued attribute can be linked to an entity or a relation node, but it might not be required. The purple one-way arrows indicate a "must_connect" relation. E.g., a relation node must be associated with two entities. As another example, an *ISA* representation needs to connect to at least two entity nodes, to demonstrate the hierarchy relations between them. The block property can be used across problems within a domain. Figure 6.5 shows a simple ER diagram that models a personnel database. We can use this sample ER diagram to illustrate how the block property ontology in Figure 6.4 would apply during the block property check. Three entities are defined (shown in blue rectangles), where *Employee* has three attribute nodes: *Salary, Name*, and *ID*, *Department* has one attribute *DName*, and *Manager* inherits properties from *Employee* through an *ISA* relationship. Two relations, *WorksIn* and *ManageBy* are defined to connect the three entities. We can see in Figure 6.5 that each attribute node is associated with one entity, and entities are linked through either inheritance or specific relations, which satisfies the requirements from Figure 6.4. If an attribute node, e.g., the *ID* node in entity *Employee*, is shared by another object *WorksIn*, it would violate the block property ontology, which says an attribute node can have only one connection.

When the evaluation starts, it iterates through each block in the diagram, and retrieves its relevant requirements from the block property ontology. These requirements are used to determine whether the given block has a valid representation.

6.3.3.3 Problem-specific check

The third step, after the general property check and the block property check, is to compare whether the given diagram contains similar information as the expert's diagram, as a solution to the given problem. It is problem-specific and is based on the expert solution. Below we list seven items that would be valuable in revealing information from a diagram. *BLOCK* and *CONNECTION* classes and their related ones would be used to represent these items. The seven items contain diagram information from two perspectives: 1) information about objects (*BLOCK*): *critical blocks, supplemental blocks, number of blocks*; and 2) information about connectivity among objects (*CONNECTION*): *critical connections, supplemental connections, conceptual connections* and *number of connections*. In particular, objects are tagged in two types: critical and supplemental, according to the importance of the objects in conveying knowledge in a diagram. Domain experts can label objects as critical or supplemental when authoring the expert solution. Or, if an object's representation can reveal its identity, the tutoring system could automatically classify an object's importance.

Critical blocks- blocks that have the highest priority to be checked. They are usually the fundamental and primary elements in a diagram. They will all be included in the expert solution. If a block's identity can be inferred from its representation shape, by searching domain knowledge and problem input, the tutoring system can automatically identify critical blocks from the expert solution. Otherwise, by default, all the blocks that appear in expert solution are critical. Or, the expert can specify specific blocks as critical when authoring the problem.

Critical connections- important connections between critical blocks. They should all be included in expert solution. Again, if a block's identity can be inferred from its representation shape, the system can automate the search process by finding all the connections between critical blocks. Otherwise, by default, all the connections appearing in the expert solution are critical. Or, the expert needs to specify them when authoring the problem.

Supplemental blocks- blocks that receive a secondary check. They are the complement of *critical blocks*, i.e., all blocks that are not critical. They are less important than the critical blocks since they are not the backbone of the diagram, i.e., they alone cannot make meaning but rather have to be attached to the critical blocks. Supplemental blocks can extend the information and provide more details about the critical blocks. Again, if a block's identity can be inferred, the tutoring system should automatically detect the supplemental blocks. Otherwise, by default, it has a null value (no blocks are supplemental). If the expert, while authoring, designates any blocks as critical, then all other blocks become supplemental. Or, the expert can specifically designate a block as supplemental.

Supplemental connections- connections between supplemental blocks, or between supplemental blocks and critical blocks. By default, it has a null value if a block's identity cannot be inferred (no connections are supplemental). The expert can also manually specify them, too.

Number of blocks- the number of blocks included in expert solution.

Number of connections- the number of connections included in expert solution.

Conceptual connections- grouped connections that together convey a concept. Conceptual connections are not additional connections, but a subset from *critical connections* and *supplemental connections*. They are defined by the expert author to enable the tutor to give specific feedback clarifying conceptual knowledge. If a block's identity can be inferred from its shape, the tutoring system will combine domain knowledge and the block's identity to group connections with shared properties. For instance, in ER diagrams, connections from an entity (shown as a rectangle) to its attributes (ellipses) can be grouped to represent a single concept: *entity-attribute*. In another example, if an ER diagram requires an employee ID as a key object, the connection between an employee entity and the ID attribute is a conceptual connection.

These conceptual connections are important pedagogically to identify conceptual misunderstandings. If student's diagram misses any connection or has additional connection between entity and attribute, it will fail on conceptual connection check. Another example, in process flow diagram, a diamond shape represents a decision-making process. Related connections on this block can reveal the student's knowledge on how to make a decision in such circumstance. So if any error is detected among connections to/from decision-making object, we think a decision-making misconception is found. The mapping between representations and conceptual connections can be found in the ontology of each type of block diagram. In addition, the expert can manually define conceptual knowledge among connections, if the shape of a block does not imply its identity. Conceptual connections can have several categories. The expert can specify them by defining each one as an individual concept, which relates to a set of connections.

The seven items of the problem-specific check listed above can apply to any type of Block Diagram. Specifically, every diagram will have a value for *Number of blocks* and *Number of connections*. Depending on the applications, some of the seven items might have a null value. E.g., if experts don't map domain concepts to connections in the diagram, *conceptual connections* would be an empty set. Another example is that in a concept map, the block representation doesn't reveal its conceptual meaning. According to our definition, by default, all blocks appearing in the expert solution will be critical. Thus, *supplemental blocks* will have a null value.

6.3.4. Pedagogy in Block Diagrams

One feature of our method that distinguishes it from other systems that simply analyze diagrams, is that it gives instructional feedback in response to errors in the diagram and helps the student move forward. Pedagogical instruction is given based on the output of the evaluation process. As we stated before, there are three evaluation steps: 1) General property check, 2) Block property check, and 3) Problem-specific check. Instructional feedback attempts to resolve each type of error one at a time (Figure 6.3).

Before we start to discuss our pedagogy approach, we would like to mention some earlier work on Diagram-based ITSs using constraint-based modeling (CBM). Unlike rule-based modeling to model both the correct solution path and the buggy ones, CBM uses set of constraints to capture domain knowledge, where each constraint represents a small section of the domain. The instructional feedback is bounded with each constraint, so that the violation of a constraint triggers feedback to a student. Without a cognitive model, CBM can provide feedback without knowing what causes the violation of the constraint. When using systems that can generate multiple possible feedback messages simultaneously, the tutor architect has to resolve the problem of what to do in this situation: i.e., if a few constraints are violated, should the tutor generate a feedback message by concatenating feedback messages from all the violated constraints, or only pick one with the highest priority? Also, bounding feedback with constraints is less flexible in terms of modularity. E.g., different domain experts might give different feedback whereas constraints are stable with a domain. To update a feedback, it is unnecessary to change the constraint. Thus, for the domain expert who has no programming experience, a separate pedagogical module that is able to generate feedback based on structure differences is highly desired. Structure difference is a concept we adopted from Py et al. (2008), which states the difference between expert solution and student's diagram in terms of connectivity among objects. The components within the pedagogical module, i.e., types of errors, and feedback messages addressing each error, should be accessed and updated by the domain expert easily without significant programming work. Based on Py et al. (2008), we developed a pedagogy taxonomy to handle different types of errors, based on the structure differences between expert's and student's solution. However, the structure differences taxonomy in Py's work focuses on one particular domain, UML, which cannot be easily generated to other domains. In order to benefit all possible diagrams in the Block Diagram family, we propose a more general and easier-to-manipulate approach to map pedagogical feedback to diagnosis output.

Feedback from general property check

Feedback on overall appearance will address any error from general property check outputs. If several errors are found in the general property check, the feedback message delivered will concatenate feedback from all the errors. For instance, if a student's process flow diagram has two isolated groups of blocks, and undirected lines as connection type, the system would decide the diagram violates two properties: connectivity and flow. The resulting feedback would prompt, "Consider that a process flow diagram requires any block to be reachable from other blocks. In order to demonstrate the flow, you need to use arrows to indicate the direction of flow."

Feedback from block property check

The block property check gives feedback on each problematic block representation. For instance, if entity node (a rectangle) is connected to another entity node, which is not permitted in ER diagram, evaluation engine will send *entity-check-fail* notion to pedagogy model. If more than one block representations fail, instruction composition would concatenate feedback of all failed types of blocks. For example, if two entity blocks are failed, only one message tackling representation properties in entity block will be provided. However, if both an entity and an attribute block fail the check, instructional message will include how to represent entity and attribute object in ER diagram.

Feedback from problem-specific check

To define problem-specific errors, we used Py et al.'s definitions (2008) and summarized with five categories: omission, insertion, replacement, reversion of direction and misconception. These five categories of errors are described below.

a). Omission error: if a student’s diagram has fewer blocks than the expert-defined *critical blocks.* To check if a student’s diagram is missing any blocks, the system compares all blocks in the collection. If any block is missing, feedback is triggered to prompt the student about the omission of the block. Note that the detail of the feedback depends on the expert’s pedagogical strategy and preference. E.g., an expert could prompt the student with the details of the missing block, or mention only that some block is omitted without indicating which one.

b). Insertion error: if the student’s diagram has more blocks than expert-defined *critical blocks* plus *supplemental blocks.* To evaluate whether a student’s drawing has extra blocks, the system goes through blocks from critical blocks to supplemental blocks as defined by the expert. If any block is outside the collection, an error will be triggered, and feedback on how to fix insertion error will be generated by the pedagogy module.

c). Replacement error: if the student’s diagram has the correct number of connections and number of blocks, but some blocks in the expert diagram are replaced with incorrect blocks. In practice, this error can be interpreted as the student’s diagram has the same *number of connections* and *number of blocks* as the expert solution, but with different collections of *critical blocks* and *supplemental blocks*. For example, if the expert solution is shown in Figure 6.6 (a), while the student drawing is shown in (b). The student’s diagram has the same number of blocks (3) and same number of connections (2) as the expert’s diagram. However, it contains a block D, which is not shown in expert solution. Therefore, it satisfies the definition of a replacement error. Furthermore, the student’s diagram may have different linkage arrangements from expert solution. As is also shown in Figure 6.6 (c, expert) and (d, student), the student and expert’s drawings have same number of blocks (5) and same number of connections (4). However, block B has 3 connections in expert diagram, whereas it has only 2 connections in student drawing. Still, this configuration satisfies our definition of replacement error.

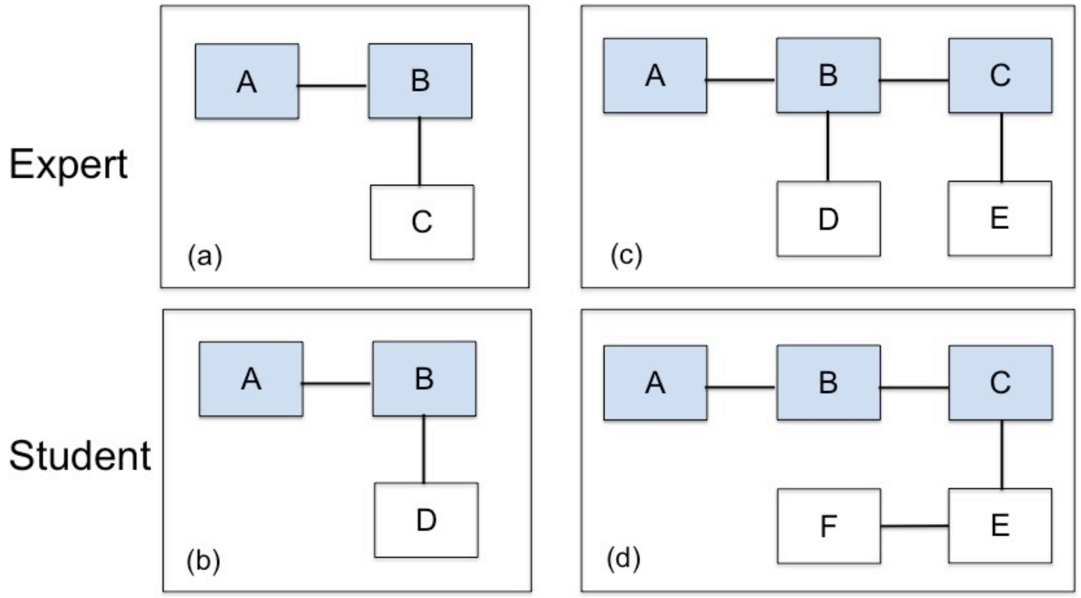


Figure 6.6. Block Diagram examples to illustrate replacement error type. (a) and (c) are expert solutions. (b) and (d) are student solutions. The blocks highlighted by blue colors are critical blocks.

d). Reversion of direction error: a connection has a reversed relation between two blocks in student's diagram. It depends on evaluation output from *critical connections* and *supplemental connections*. If any connection has a reversed direction link, feedback will be triggered for this error.

e). Misconception error: if student's diagram fails on the *conceptual connection* check. As described above, conceptual connections are grouped connections that together convey a concept. The system automatically detects conceptual connections from the expert solution by combing domain knowledge and a block's identity if it is inferable from its shape. Therefore, feedback generated by misconception error will be the same across problems in a particular domain. However, if conceptual connections are not available in a particular problem, the misconception node will be ignored in the pedagogy module.

The two sections 6.3.3 and 6.3.4 we discussed above can be organized in an ontology showing in Figure 6.3. In particular, the yellow nodes denote the three evaluation steps (Section 6.3.3.1 to 6.3.3.3). Also, it includes classes (green notes) we defined in Block Diagram in Section 6.3.2 to relate class representation with evaluation elements. Pedagogy feedback to correct errors in each evaluation step is shown in orange nodes. Blue arrows are used to model the relations between error nodes and different levels of information in the diagram, which have been discussed in Section 6.3.4.

Let us walk through an example of using the ontologies with a new diagram. To use the evaluation/pedagogy ontology (Figure 6.3), two processes are involved. First, given a diagram, its diagnosis should follow the three-step evaluation process. In particular, a function is tied to each property defined in general propety ontology (Figure 6.4). Thus, all relevant functions are enabled in general property check. Pedagogical instructions are prepared for each property. In the block property check step, the representation check (yellow node) will be processed by being directed to the block property ontology (see an example in Figure 6.5). By iterating through each block, its number of connections and connecting types will be validated. Pedagogical feedback should be prepared for each block type. In problem-specific check step, the system will parse the diagram and build internal representation of the diagram information by filling the values/instances in the white ovals. In this step, pedagogical instructions for five types of errors should be provided. Since the error type is generic, that works across any Block Diagram, and ideally, the pedagogical instructions will be re-used to across domains.

Even though the above process is somewhat complex, it is systematic and well-defined. Without these ontology-based tools, a domain expert wanting to evaluate a new diagram would need to start from scratch. The first step would be defining the components of the diagram, and the next would be specifying the order of evaluation steps. Because different experts have different preferences, the results of these decisions might vary significantly without a tool like the above ontologies to guide them. Also, using these tools enables more significant time savings when applying them to multiple problems within a domain, because the newly adapted ontologies can be re-used for each new problem.

6.4. Applications in Three Domains

In this section, we choose three types of diagrams from the five subcategories in Block Diagram to demonstrate the application of our knowledge representation design and evaluation ontology work. The ontologies are applied to process flow diagrams, ER diagrams, and circuit diagrams.

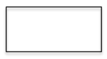
6.4.1. Process flow diagram

The process flow diagram is commonly used in engineering to indicate the general flow of processes and equipment. It contains a set of elements indicating possible materials, data, conditions, or operations. Note color can be a visual tool to group objects but is not considered here.

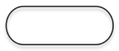
6.4.1.1. Representations

Below shows a list of representation classes widely used in process flow diagram. As an explanation, we only pick five. There are other kinds of representations in process flow diagram.

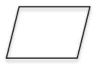
PROCESS- It describes a procedure or activity.

- Shape: 
- Identity: process
- ARandomTerminal: any point at the periphery of the shape

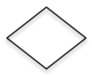
TERMINATOR- It indicates the beginning or ending of a process and link to other related process.

- Shape: 
- Identity: terminator
- ARandomTerminal: any point at the periphery of the shape

DATA- It defines data that are required to support a process step.

- Shape: 
- Identity: data
- ARandomTerminal: any point at the periphery of the shape

DECISION- It describes a decision point in the process and the subsequent process steps that depend on which option is selected. It has at least three connections, with one entry and two exists.

- Shape: 
- Identity: decision
- ARandomTerminal: any point at the periphery of the shape
- AVertexTerminal: any point at the vertices of the shape

CONNECTOR- It represents converging paths. It has at least two entries, but one exit.

- Shape: ○
- Identity: connector
- ARandomTerminal: any point at the periphery of the shape

<u>*6.4.1.2. Process flow diagram evaluation*</u>

1) General property check. Based on Figure 6.4, it should include *integrity, connectivity* and *flow check*.

2) Block property check. Based on domain knowledge, here we build an ontology to illustrate legal connections between each type of representation (Figure 6.7). Basically, there is no restriction on the maximum number of connections on each type, and they are allowed to connect to any type on the representation list. According to the definition, the *PROCESS, TERMINATOR* and *DATA* should have at least one connection, and the *DECISION* and *CONNECTOR* should have at least three connections.

3). The problem-specific check will include the components we defined before. Please check Figure 6.3. Note it defines one more relation between representation node and problem-specific check. I.e., the decision block is related to *decision making,* which is a type of conceptual connection. In other words, connections entering or exiting from the decision block will be considered to be conceptual connections. If a student makes an error on connections from a decision block, he/she would fail the conceptual connections check. An example of a process flow diagram and its evaluation ontology will be discussed in Section 6.6 in Case Study 1.

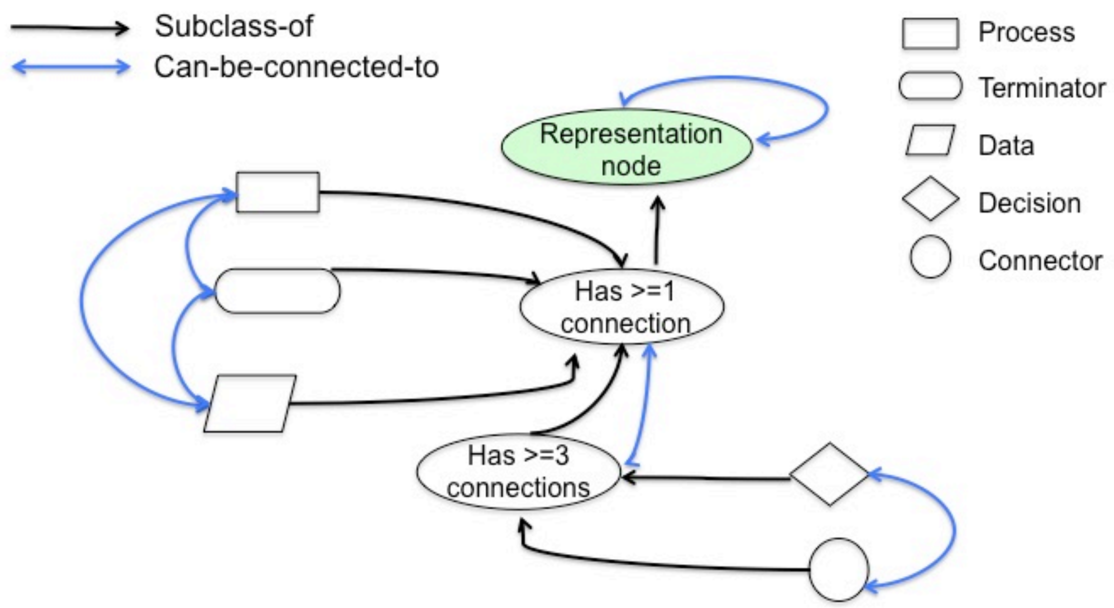


Figure 6.7. Block property ontology in process flow diagrams.

6.4.2. ER diagram

As a second example domain, in software engineering, the Entity-Relation diagram is used to describe the data or information aspects of a business domain or its process requirements, which will later be implemented in a database. The main components of an ER diagram are entities and relationships among them.

6.4.2.1. Representations

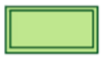
Below is a list of representation classes commonly used in an ER diagram. Note that the color associated below with each shape of representation will not affect the meaning. Color is used in this document as a way to distinguish and highlight each type.

ENTITY- It describes a subject relevant to a system and can be a person or an object.

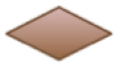
- Shape:
- Identity: entity

- ARandomTerminal: any point at the periphery of the shape.

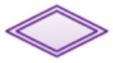
WEAK ENTITY- It is an entity that cannot be identified by its own attributes. Instead, it depends on the existence of another entity. E.g., The *order item* entity will be meaningless without an *order* entity. Thus, its primary key should contain a foreign key as well as its attributes.

- Shape: 
- Identity: weak entity
- ARandomTerminal: any point at the periphery of the shape.

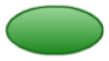
RELATION- It describes how entities interact. (It can have many terminals, and the location is not fixed. But it is traditionally better to have connections at the vertices.)

- Shape: 
- Identity: relation
- AVertexTerminal: a point from the four vertices. By convention, this is used to connect any entity node.
- ARandomTerminal: any point at the periphery of the shape.

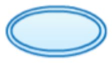
WEAK RELATION- It defines how an entity interacts with weak entity.

- Shape: 
- Identity: weak relation
- AVertexTerminal: a point from the four vertices. By convention, this is used to connect any entity node.
- ARandomTerminal: any point at the periphery of the shape.

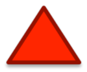
ATTRIBUTE- It is a property or a characteristic of an entity or relation. Therefore, an attribute cannot have its own attributes. It cannot exist by itself either.

- Shape: 
- Identity: attribute
- ARandomTerminal : any point at the periphery of the shape.

MULTIVALUED ATTRIBUTE- It models an attribute that can have more than one value. One example is that multiple subject values could be associated with a *teacher* entity.

- Shape: 
- Identity: multivalued attribute
- ARandomTerminal: any point at the periphery of the shape.

ISA- It models the inheritance relation between two entities. E.g., a *student* entity ISA *person* entity.

- Shape: 
- Identity: Is-a relation
- ABaseTerminal(Point): a point from the lower side. By convention, this is used to connect child entity node.
- ANonBaseTerminal(Point): any point from the two inclined sides. By convention, this is used to connect parent entity node.

<u>6.4.2.2. Conceptual connections</u>

Here we define three types of connections as conceptual connections in an ER diagram.

Entity-relation: connections between an *ENTITY* and a *RELATION*. This type models how an entity participates in a relation.

Entity-attribute: connections between an *ENTITY* and an *ATTRIBUTE*. This type models properties within an entity.

Entity-inheritance: connections between two *ENTITY* nodes. In other words, it models a parent/child node relationship.

6.4.2.3. ER diagram evaluation

1) General property check. Based on Figure 6.4, it should include *uniqueness, connectivity*, and *integrity* check.

2) Block property check. Based on definition of each representation class, we build an ontology to specify legal connections among them. The block property ontology is shown in Figure 6.4.

a. Any *ATTRIBUTE* or *MULTIVALUED ATTRIBUTE* should have exactly one connection to either an *ENTITY* or *WEAK ENTITY* or to a *RELATION* or *WEAK RELATION*, since based on the definition, they cannot be shared by different objects.

b. Any *ENTITY* or *WEAK ENTITY* should have at least one connection. We don't set up an upper bound at this stage, however.

c. Any *RELATION* or *WEAK RELATION* should have at least two connections. According to their definitions, they are used to connect an *ENTITY* or *WEAK ENTITY*.

3) Problem-specific check. Again, it has the same components discussed in Figure 6.3. For conceptual connections, we define three sub-types: *Entity-relation, Entity-attribute* and *Entity-inheritance*. Each of them is related to some representation classes. If a student's drawing fails on any one of them, the corresponding misconception will get

activated. For instance, if student incorrectly put a link between an *ISA* and an *ATTRIBUTE* node, the system would decide the student has misconceptions on both *entity-inheritance* and *entity-attribute* concepts. Two ER diagram examples will be discussed in Section 6.6 in Case Studies 2 and 3.

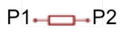
6.4.3. Circuit diagram

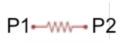
A third type is circuit diagram. A circuit diagram or electrical diagram is a graphical representation of an electrical circuit. A schematic diagram shows the components and connections of the circuit using standardized symbolic representations. Symbols are used to represent the features of the physical device.

6.4.3.1. Representations

Below shows a list of representation classes widely used in circuit diagram. Once they are defined, they can be re-used across problems.

RESISTER

Shape: 



Identity: resister

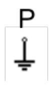
Terminals: Tl_1, Tl_2

Loc_of_Terminals: (Tl_1, P1), (Tl_2, P2),

Num_of_Terminals: 2

Equiv_Terminals: <Tl_1, Tl_2>

EARTH GROUND

Shape: 

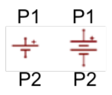
Identity: earth ground

Terminals: Tl

Loc_Terminal: (Tl, P)

Num_of_Terminals: 1

BATTERY

Shape: 

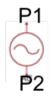
Identity: battery

Terminals: Tl_pos, Tl_neg

Loc_of_Terminals: (Tl_pos, P1), (Tl_neg, P2)

Num_of_Terminals: 2

AC VOLTAGE SOURCES

Shape: 

Identity: AC voltage

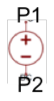
Terminals: Tl_1, Tl_2

Loc_of_Terminals: (Tl_1, P1), (Tl_2, P2)

Num_of_Terminals: 2

Equiv_Terminals : <Tl_1, Tl_2>

DC VOLTAGE SOURCES

Shape: 

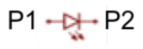
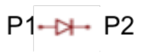
Identity: DC voltage

Terminals: Tl_pos, Tl_neg

Loc_of_Terminals: (Tl_pos, P1), (Tl_neg, P2)

Num_of_Terminals: 2

LED/DIODES

Shape:  

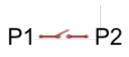
Identity: diode

Terminals: Tl_pos, Tl_neg

Loc_of_Terminals: (Tl_pos, P1), (Tl_neg, P2)

Num_of_Terminals: 2

SWITCH

Shape: 

Identity: switch

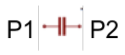
Terminals: Tl_1, Tl_2

Loc_of_Terminals: (Tl_1, P1), (Tl_2, P2)

Num_of_Terminals: 2

Equiv_Terminals: <Tl_1, Tl_2>

CAPACITOR

Shape: 

Identity: capacitor

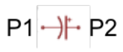
Terminals: Tl_1, Tl_2

Loc_of_Terminals: (Tl_1, P1), (Tl_2, P1)

Num_of_Terminals: 2

Equiv_Terminals : <Tl_1, Tl_2>

POLARIZED CAPACITOR

Shape: 

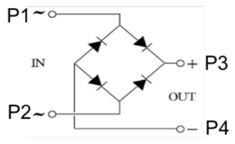
Identity: polarized capacitor

Terminals: Tl_pos, Tl_neg

Loc_of_Terminals: (Tl_pos, P1), (Tl_neg, P2)

Num_of_Terminals: 2

BRIDGE RECTIFIER

Shape: 

Identity: bridge rectifier

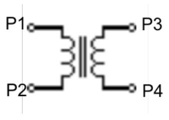
Terminals: Tl_ac_1, Tl_ac_2, Tl_pos, Tl_neg

Loc_of_Terminals: (Tl_ac_1, P1), (Tl_ac_2, P2), (Tl_pos, P3), (Tl_neg, P4)

Num_of_Terminals: 4

Equiv_Terminals : <Tl_ac_1, Tl_ac_2>

TRANSFORMER

Shape: 

Identity: transformer

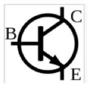
Terminals: Tl_low_1, Tl_low_2, Tl_high_1, Tl_high_2

Loc_of_Terminals: (Tl_low_1, P1), (Tl_low_2, P2), (Tl_high_1, P3), (Tl_high_2, P4)

Num_of_Terminals: 4

Equiv_Terminals : <Tl_low_1, Tl_low_2>, <Tl_high_1, Tl_high_2>

NPN BIPOLAR TRANSFORMER

Shape: 

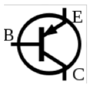
Identity: NPN bipolar transformer

Terminals: Base, Emitter, collector

Loc_of_Terminals: (Base, B), (Emitter, E), (Collector, C)

Num_of_Terminals: 3

PNP BIPOLAR TRANSFORMER

Shape: 

Identity: PNP bipolar transformer

Terminals: Base, Emitter, collector

Loc_of_Terminals: (Base, B), (Emitter, E), (Collector, C)

Num_of_Terminals: 3

6.4.3.2. Circuit diagram evaluation

1). General property check. According to Figure 6.4, it includes *connectivity* and *integrity* check.

2). Block property check. Based on the definition of each representation, we build an ontology to illustrate legal connections among them. As is shown in Figure 6.9, the representation classes are categorized into four sub-classes, based on their allowable connections. For instance, *EARTH GROUND* can have only one connection, whereas *TRANSFORMER* must have four connections by linking two *AC VOLTAGE SOURCE*. The blue double arrow shows a *can-be-connected* relation, whereas the orange arrow indicates a *need-to-be-connected* relation, based on the conceptual level of each object.

Note that the information in this block property ontology is not complete. The purpose of showing it is simply to illustrate a way that a block property can be represented.

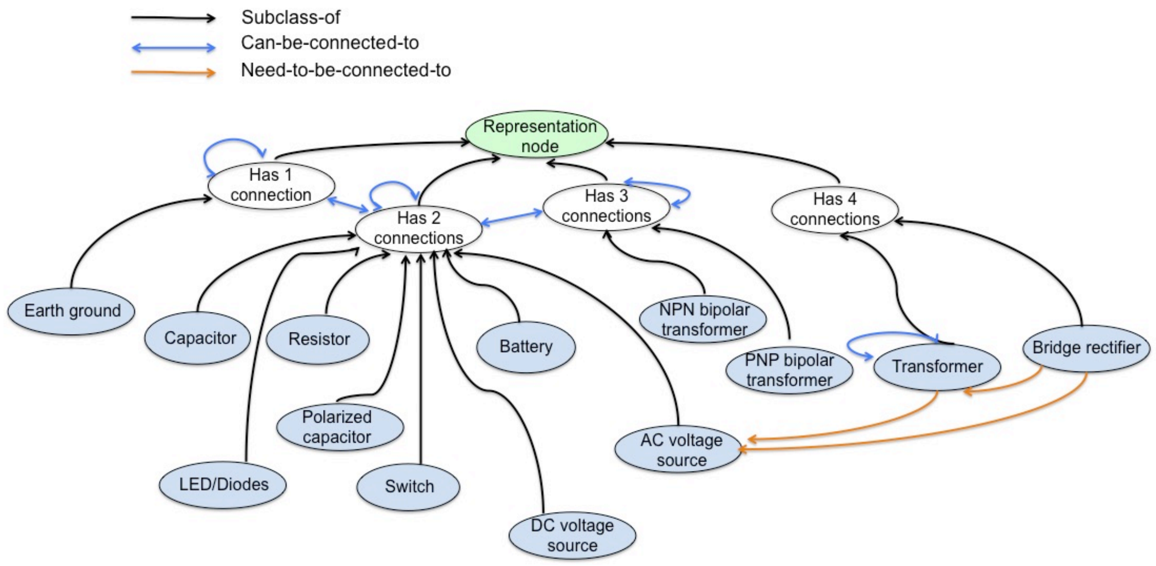


Figure 6.8. Block property ontology in circuit diagrams.

3). Problem-specific check. It follows the rules defined in Figure 6.3. Even though the shape of a block reveals its identity, without having problem-specific information, the system cannot identify critical blocks automatically. In conceptual connections, we define two sub-classes: *Rectifier* and *AC voltage transform*. The definition of rectifier is to change AC to DC. So if student's drawing fails on connections from *BRIDGE RECTIFIER*, the system will decide he/she has a misconception on how a rectifier should work. Another concept is *AC voltage transform*, where connections from a transformer in the diagram can depict it.

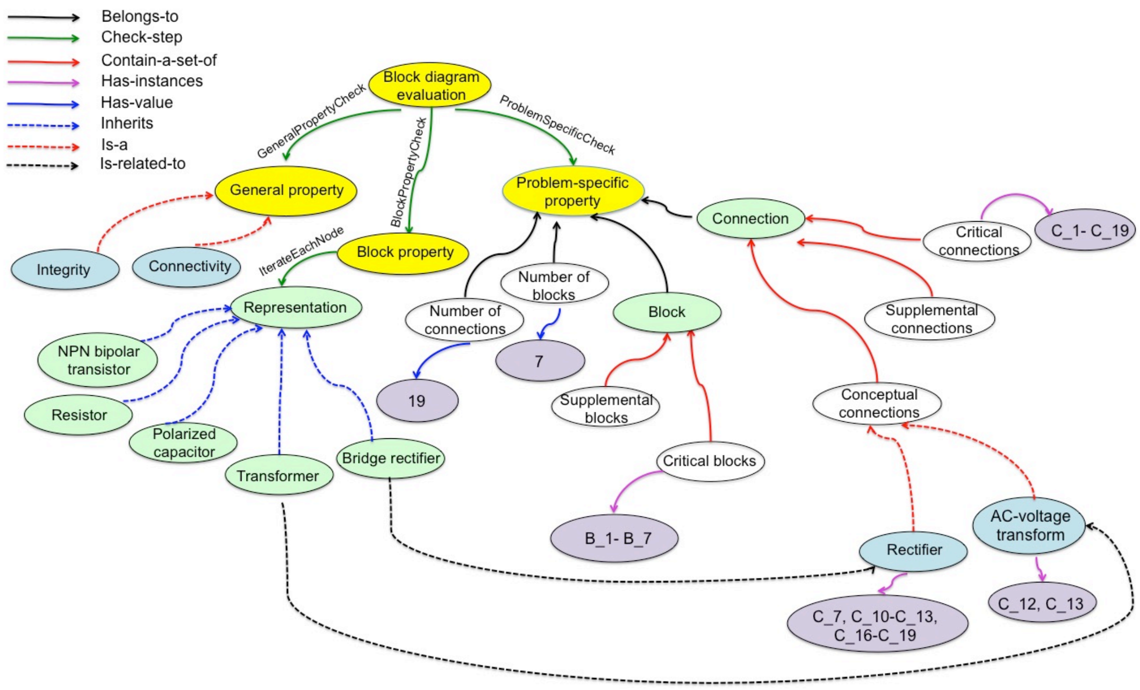


Figure 6.9. Evaluation process ontology in a circuit diagram. Yellow nodes: 3-step evaluation process. Note, the general property ontology (Figure 6.4) is required to retrieve diagram's properties for general property check. The block property check for each representation class will be forwarded to a domain-dependent block property ontology (Figure 6.9). Green nodes: abstract classes that are domain-dependent. White nodes: data structures that contain information from a diagram. Purple nodes: information/instances retrieved from an expert diagram. Detailed information is shown in Tables 6.1 and 6.2. Blue nodes: general property or conceptual connection subtypes that belong to a given domain. Across-domain nodes: yellow and white; Domain-dependent nodes: blue and green; problem-dependent nodes: purple.

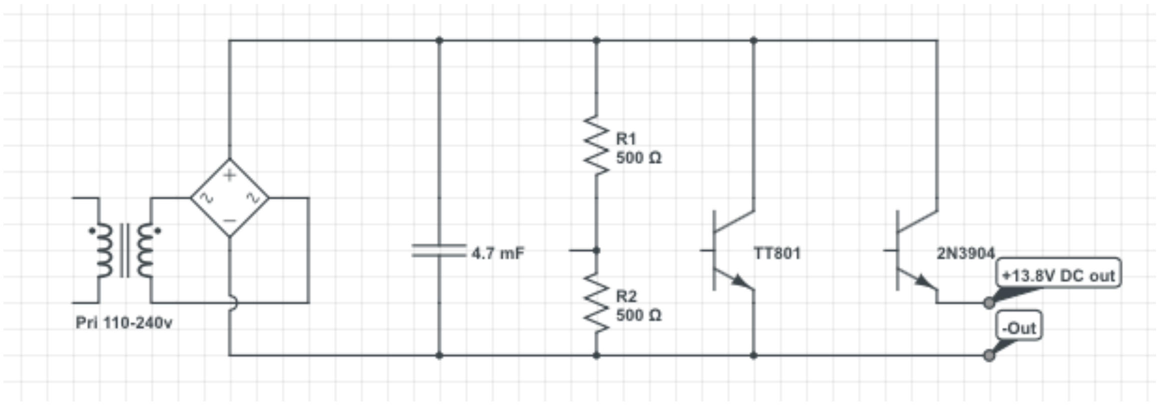


Figure 6.10. A circuit diagram example.

6.4.3.3. An example

Here, we use an example (Figure 6.10) to demonstrate knowledge representation in problem-specific check. The relevant information is listed below, and its corresponding evaluation ontology is shown in Figure 6.9.

Number of Blocks: 7

Number of Connections: 19

Set of Blocks: see Table 6.1.

Table 6.1. Set of *BLOCKs* in circuit diagram (Figure 11).

Block	**Content**->label	**Representation**	**Operation**
B_1	2N3055	**NPN bipolar transistor**	Label: 13.8 VDC out Terminal: P3
B_2	TT801	**NPN bipolar transistor**	N/A
B_3	560 Ω	**Resistor**	N/A
B_4	560 Ω	**Resistor**	N/A
B_5	4.7 mF, 4700μF	**Polarized capacitor**	Label: -out Terminal: P2
B_6	N/A	**Bridge rectifier**	Label: -out Terminal: P4
B_7	Pri 110-240v ~ sec 16-20v~	**Transformer**	N/A

Set of Connections: see Table 6.2.

Critical Blocks: B_1 – B_7

Critical Connections: C_1- C_19

Supplemental Blocks: None

Supplemental Connections: None

Conceptual Connections:

Rectifier: C_7, C_10, C_11, C_12, C_13, C_16, C_18, C_19

AC voltage transform: C_12, C_13

Table 6.2. Set of *CONNECTIONs* in circuit diagram (Figure 11).

Connection	Node_1	Node_2	Label	Connecting Type	Point_1	Point_2
C_1	B_1	B_2	N/A	Line	B_1-> P1	B_2 ->P2
C_2	B_1	B_2	N/A	Line	B_1-> P3	B_2 ->P3
C_3	B_2	B_4	N/A	Line	B_2 -> P3	B_4 ->P1
C_4	B_2	B_4	N/A	Line	B_2 ->P1	B_4 ->P2
C_5	B_1	B_4	N/A	Line	B_1 -> P2	B_4 ->P1
C_6	B_1	B_5	N/A	Line	B_1 ->P3	B_5 ->P1
C_7	B_1	B_6	N/A	Line	B_1 -> P2	B_6 ->P3
C_8	B_3	B_5	N/A	Line	B_3 ->P2	B_5 ->P2
C_9	B_2	B_3	N/A	Line	B_2 -> P1	B_3 ->P1
C_10	B_5	B_6	N/A	Line	B_5-> P2	B_6 -> P4
C_11	B_5	B_6	N/A	Line	B_5 -> P1	B_6 ->P3
C_12	B_6	B_7	N/A	Line	B_6 ->P1	B_7 ->P1
C_13	B_6	B_7	N/A	Line	B_6 -> P2	B_7->P2
C_14	B_4	B_5	N/A	Line	B_4->P1	B_5->P1
C_15	B_3	B_4	N/A	Line	B_3 ->P1	B_4 ->P2
C_16	B_4	B_6	N/A	Line	B_4->P1	B_6 ->P3
C_17	B_2	B_5	N/A	Line	B_2 ->P3	B_5 ->P1
C_18	B_3	B_6	N/A	Line	B_3 ->P2	B_6 ->P4
C_19	B_2	B_6	N/A	Line	B_2 ->P3	B_6 ->P3

6.4.4. Summary

In this section, we have demonstrated that our knowledge representation and evaluation process are applicable to three different domains. In particular, we provided class representation, block property and evaluation ontologies for process flow diagrams, ER diagrams and circuit diagrams. Examples of the process flow diagram and ER diagram will be discussed in our case studies in Section 6.6. In the next section, we will continue to apply our ontologies in another domain that doesn’t use specific geometric shape to represent the conceptual information.

6.5. Other Application: Concept Map

To demonstrate that our ontologies can be applied to a more general type of diagram, in which block shape doesn’t contribute to the meaning of the content, we test it with concept map. A concept map or conceptual diagram is a diagram that shows a suggested relationship between concepts. It is a graphical tool that supports organize and structure knowledge (Hager, Scheiber, & Corbin, 1997). In general, information is represented as boxes or circles, with labeled arrows in a downward-branching hierarchical structure. Linking phrases can be used to articulate relationships between concepts.

6.5.1. Evaluation ontology

Based on general property ontology in Figure 6.4, the concept map belongs to the inheritance/hierarchy subfamily, where parent-child relations need to be denoted by using arrows to link the blocks. By inheriting properties from all of its upper-level categories, it should have four attributes: *connectivity, integrity, uniqueness* and *parent-child relation*, shown by the blue nodes at the left branch in Figure 6.12.

In a concept map, the shape of the block doesn’t reveal specific meaning; only the content inside the block matters. So the block property check will be skipped. (See the middle branch was eliminated in Figure 6.12.)

Again, the problem-specific check follows the components we defined in Figure 6.3.

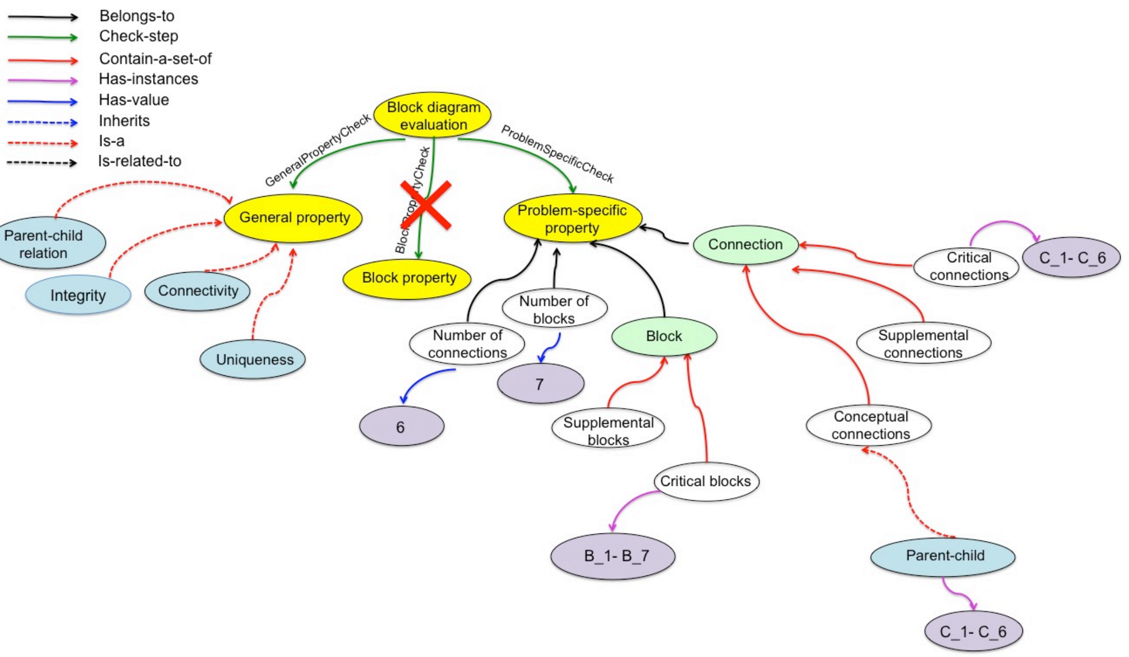


Figure 6.11. Evaluation process ontology in a concept map. Only general property check and problem-specific check are defined. Yellow nodes: 3-step evaluation process. Note, the general property ontology (Figure 6.4) is required to retrieve the diagram's properties for general property check. Green nodes: abstract classes that are domain-dependent. Note representation classes are not defined. White nodes: data structures that contain information from a diagram. Purple nodes: information/instances retrieved from an expert diagram. Detailed information is shown in Tables 6.3 and 6.4. Blue nodes: general property or conceptual connection subtypes that belong to a given domain. Across-domain nodes: yellow and white; Domain-dependent nodes: blue and green; problem-dependent nodes: purple.

6.5.2. An example

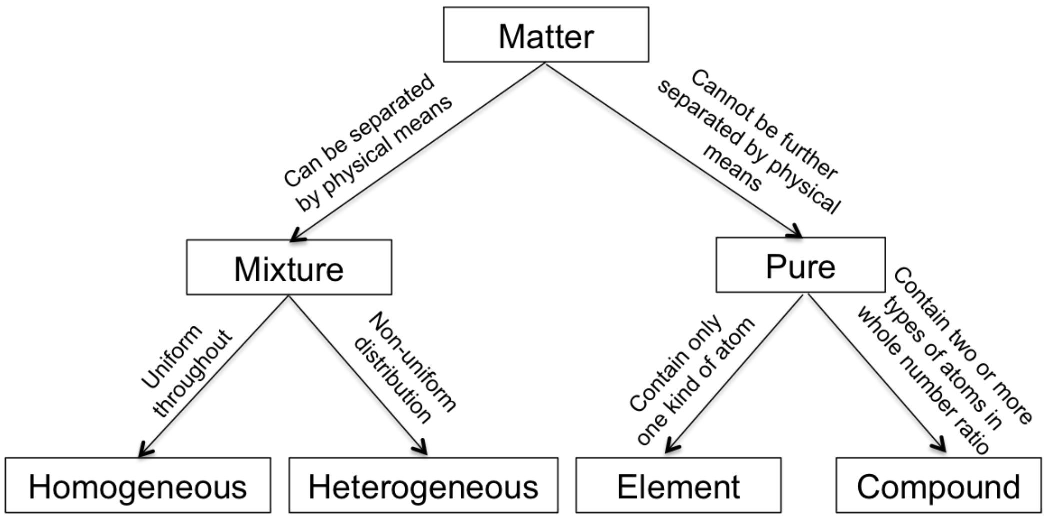

Matter
Can be separated by physical means
Cannot be further separated by physical means
Mixture
Pure
Uniform throughout
Non-uniform distribution
Contain only one kind of atom
Contain two or more types of atoms in whole number ratio
Homogeneous
Heterogeneous
Element
Compound

Figure 6.12. A concept map in chemistry: categories of matter.

As an example, assume Figure 6.12 is the expert solution. After parsing the diagram, the system should maintain a set of information, which is shown below. The information will be used in problem-specific check step, where student's diagram will be converted to similar format and compared with the expert's knowledge. Figure 6.12 shows the corresponding evaluation ontology. A *parent-child* concept is defined to as one of the conceptual connection types.

Number of Blocks: 7

Number of Connections: 6

Set of blocks: see Table 6.3.

Set of connections: see Table 6.4.

Table 6.3. Set of *BLOCKs* in concept map (Figure 6.13).

Block	**Content**->label	**Representation**	**Operation**
B_1	Matter	N/A	N/A
B_2	Mixture	N/A	N/A
B_3	Pure	N/A	N/A
B_4	Homogeneous	N/A	N/A
B_5	Heterogeneous	N/A	N/A
B_6	Element	N/A	N/A
B_7	Compound	N/A	N/A

Table 6.4. Set of *CONNECTIONs* in concept map (Figure 6.13).

Connection	Node _1	Node _2	Label	Connecting type	StartPoint	EndPoint
C_1	B_1	B_2	Can be separated by physical means	Arrow	B_1-> ARandomTerminal	B_2 -> ARandomTerminal
C_2	B_1	B_3	Cannot be further separated by physical means	Arrow	B_1-> ARandomTerminal	B_3-> ARandomTerminal
C_3	B_2	B_4	Uniform throughout	Arrow	B_2-> ARandomTerminal	B_4-> ARandomTerminal
C_4	B_2	B_5	Non-uniform distribution	Arrow	B_2-> ARandomTerminal	B_5-> ARandomTerminal
C_5	B_3	B_6	Contain only one kind of atom	Arrow	B_3-> ARandomTerminal	B_6-> ARandomTerminal
C_6	B_3	B_7	Contain two or more types of atoms in whole number ratio	Arrow	B_3-> ARandomTerminal	B_7-> ARandomTerminal

Critical Blocks: B_1 –B_7

Critical Connections: C_1 – C_6

Supplemental Blocks: None

Supplemental Connections: None

Conceptual Connections

Parent-child: C_1 – C_6

To summarize, in this section, we applied our ontologies to a concept map, which doesn't use specific geometric shape to reveal conceptual knowledge. Compared with the previous three types in Section 6.4, it is more general and needs less implementation. Thus, we demonstrate our method is feasible to various subtypes of Block Diagram.

6.6. Validation

Now we would like to offer three case studies to demonstrate how to use the general property ontology (Figure 6.4) and evaluation/pedagogy ontology (Figure 6.3) to create ITSs to tutor student's diagrams. The three cases are prepared to demonstrate that our approach 1) is a good method to construct knowledge representation and evaluation process for a Diagram-based ITS from scratch, 2) is easy to transfer to another domain based on the existing information, and 3) is easy to create a new problem once a domain has defined.

Case Study 1. Construct an evaluation process ontology for process flow diagram from scratch.

An expert plans to create an ITS to tutor a process flow diagram (Figure 6.13) in an engineering course. The ITS should contain 1) a domain model which includes a set of rules on how each domain-defined object should connect, restrictions on number of terminals with each object 2) an expert model with an expert solution, 3) an evaluation engine that has algorithms to compare student's diagram with the expert solution, and 4) a pedagogical model which is able to generate feedback to student when an error is detected.

Without the aid of a design tool, this process is time-consuming. As we discussed in Chapter 2, the current widely used approach in tutoring diagrams is constraint-based modeling (CBM). CBM uses set of constraints to capture domain knowledge, where each constraint represents a small section of the domain. The student's solution is deemed correct only if all the constraints in the given domain are satisfied, in addition to matching the knowledge in expert solution. CBM gives student feedback only when a domain principle is violated. Therefore, to create an ITS in a process flow diagram using CBM approach, the expert has to construct a set of domain-wide constraints with feedback associated, e.g., "all the processes should be linked by directed line." A comparison between the expert and student's drawing is based on text representations by converting the graphic information to syntax-based relations. As one can notice, this is tedious work requiring significant domain expertise, software design, knowledge representation and programming.

To show the benefit of our method, we use Block Diagram evaluation ontology and general property ontology to simplify the process, since a process flow diagram is a subtype of Block Diagram.

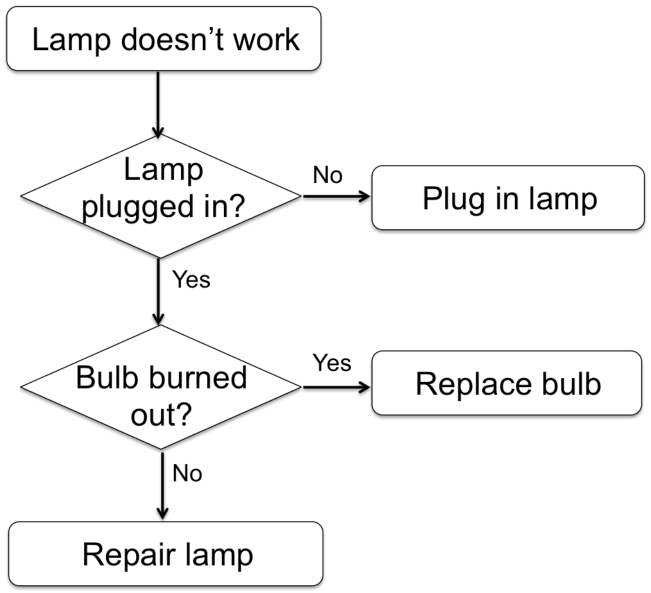


Figure 6.13. A fix-lamp process flow diagram, used in Case Study_1.

Assume there exists a UI that allows the expert to draw the diagram, i.e., a drawing palette that contains the representation objects from the specific domain. In other words, our method can understand each piece from this diagram automatically. A representation class is needed to model each representation object in the drawing palette, which will be used to create the inner computational representation of the drawing. As is mentioned in Section 6.4, we defined five classes in process flow diagram: *PROCESS, TERMINATOR, DATA, DECISION* and *CONNECTOR*.

Then our solution is to build an inner representation of this diagram by using the classes we defined. Instances of *BLOCKS* and *CONNECTIONS* are created and named (left column in Table 6.5 and Table 6.6). Note: the bolded words indicate class type.

Table 6.5. Set of *BLOCKs* in the process flow diagram.

Block	**Content**->label	**Representation**	**Operation**
B_1	Lamp doesn't work	**Terminator**	N/A
B_2	Lamp plugged in?	**Decision**	N/A
B_3	Bulb burned out?	**Decision**	N/A
B_4	Repair lamp	**Terminator**	N/A
B_5	Plug in lamp	**Terminator**	N/A
B_6	Replace bulb	**Terminator**	N/A

Table 6.6. Set of *CONNECTIONs* in the process flow diagram.

Connection	Node_1	Node_2	Label	Connecting type	StartPoint	EndPoint
C_1	B_1	B_2	N/A	Arrow	B_1-> ARandomTerminal	B_2 -> ARandomTerminal
C_2	B_2	B_3	N/A	Arrow	B_2-> ARandomTerminal	B_3-> ARandomTerminal
C_3	B_3	B_4	N/A	Arrow	B_3-> ARandomTerminal	B_4-> ARandomTerminal
C_4	B_2	B_5	N/A	Arrow	B_2-> ARandomTerminal	B_5-> ARandomTerminal
C_5	B_3	B_6	N/A	Arrow	B_3-> ARandomTerminal	B_6-> ARandomTerminal

Consulting Figure 6.3, the evaluation steps are denoted with yellow nodes, which have three branches: block diagram general property, block property and problem-specific property. We then discuss how to construct an evaluation ontology for the fix-lamp process flow diagram problem.

a) General property check. By consulting Figure 6.4, we notice that process flow diagram is in the "sequence and flow" category, which has a property node "flow." This indicates that the general property check in a process flow diagram includes *flow*, in addition to the two general features, *connectivity* and *integrity*, applicable for all block diagrams.

b) Block property check. As we mentioned previously, this step will iteratively check blocks in the diagram to check if they have a valid representation. An abstract class, *REPRESENTATION*, is defined to capture a block's properties across domains. Therefore, domain-wide classes have to be created to inherit and implement the properties from *REPRESENTATION* class. In particular, for each subclass, the expert shall specify its identity, shape and properties associated with terminals, such as number of terminals, location of terminals, equivalent terminals, and some special terminals if it is applicable. In this example, three classes are needed: *PROCESS, TERMINATOR* and

DECISION. Details about these classes in process flow diagram have been discussed in Section 6.4. The Representation oval in Figure 6.3 will be expanded by adding these three representation class nodes. Each of the representation class node will be validated by consulting block property ontology of process flow diagram (Figure 6.8).

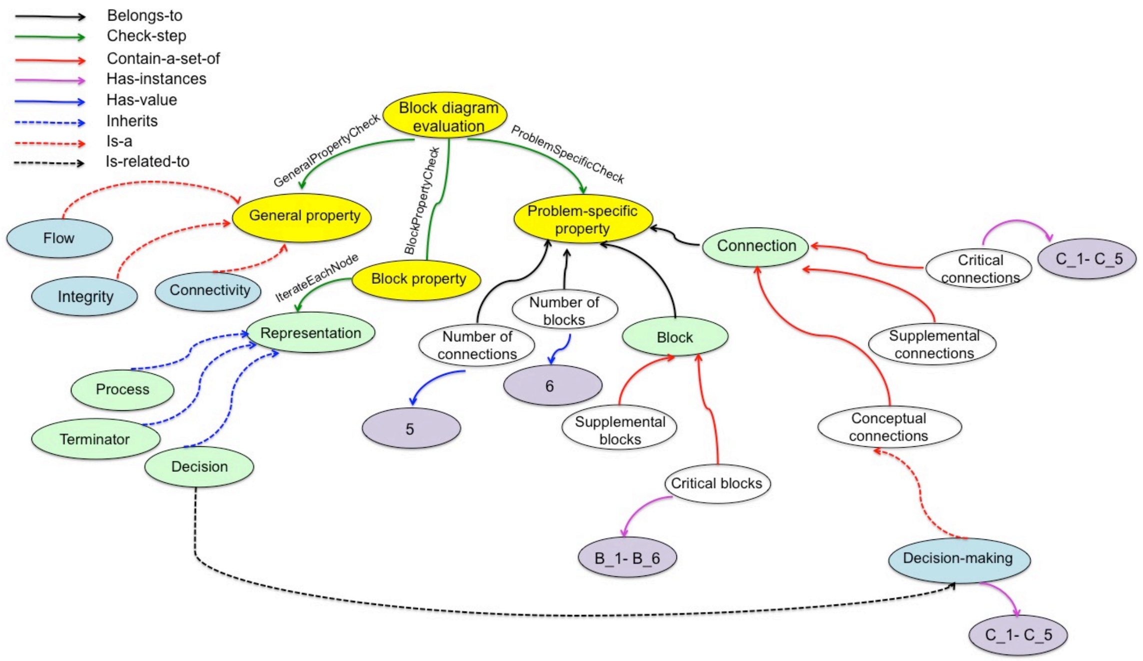


Figure 6.14. Fix-lamp process flow diagram evaluation ontologies. Yellow nodes: 3-step evaluation process. Note: the general property ontology (Figure 6.4) is required to retrieve diagram's properties for general property check. The block property check for each representation class will be forwarded to a domain-dependent block property ontology (Figure 6.8). Green nodes: abstract classes that are domain-dependent. White nodes: data structures that contain information from a diagram. Purple nodes: information/instances retrieved from an expert diagram. Detailed information is shown in Tables 6.5 and 6.6. Blue nodes: general property or conceptual connection subtypes that belong to a given domain. Across-domain nodes: yellow and white; Domain-dependent nodes: blue and green; problem-dependent nodes: purple.

c) Problem-specific check. Based on Figure 6.3, this step involves seven components: *number of blocks, number of connections, critical blocks, supplemental blocks, critical connections, supplemental connections* and *conceptual connections*, and two classes: *BLOCK* and *CONNECTION*. Information for each component can be retrieved from the diagram. Table 6.5 and Table 6.6 show a set of instances of *BLOCK* and *CONNECTION*, respectively, which can be used to fill in the purple nodes with *hasValue* or *hasInstances* relation. Categories of conceptual connections can be defined too, where any representation class can be related, shown by a blue node in right lower corner in Figure 6.15. The completed evaluation ontology is in Figure 6.15.

Furthermore, to complete the design process discussed in Figure 6.3, feedback for each pedagogy type needs to be defined by the expert.

In Case Study 1, we described an example of how to build an evaluation ontology for a process flow diagram. By comparing our method with the complexity of using a traditional method, we solve the knowledge representation issue and the design of an evaluation process in a simpler and modular way. Two assumptions were made: 1) a drawing tool UI is available for expert to pick geometric shape and draw a diagram containing these shapes and 2) the class representation for each geometric shape has been created according to our discussion in Section 6.4. Our methodology first creates instances of *BLOCKs* and *CONNECTIONs* by interpreting components in the diagram. Then the evaluation ontology is generated according to Figure 6.4 and Figure 6.3 by filling instances of classes from the example.

Case Study 2. Construct an evaluation ontology for an ER diagram, given that the evaluation ontology for process flow diagram is available from Case Study 1.

Now the expert wants to create an ER diagram tutoring system for a computer science database course (an example shown in Figure 6.5). Without any help from an existing authoring system, she needs to construct the domain, student, expert and pedagogy models from scratch. Unlike a process flow diagram, an ER diagram has different rules regarding how each object should connect and restrictions on number of terminals with each object. Also, a different approach needs to be applied to check if a student's drawing matches with an expert's diagram. Moreover, a new type of pedagogy model is desired to meet the requirements and concepts in ER modeling. As noted above, this process is painful.

Table 6.7. Instances of *BLOCKs* in personnel ER diagram.

Block	**Content**->label	**Representation**	**Operation**
B_1	Employee	Entity	N/A
B_2	Manager	Entity	N/A
B_3	Department	Entity	N/A
B_11	Salary	Attribute	N/A
B_12	Name	Attribute	N/A
B_13	ID	Attribute	N/A
B_4	isa	ISA	N/A
B_5	ManagedBy	Relation	N/A
B_6	WorksIn	Relation	N/A
B_31	DName	Attribute	N/A

Table 6.8. Instances of *CONNECTIONs* in personnel ER diagram.

Connection	Node_1	Node_2	Label	Connecting type	Point_1	Point_2
C_1	B_1	B_4	N/A	Line	B_1-> ARandomTerminal	B_4 -> ANoneBaseTerminal
C_2	B_2	B_4	N/A	Line	B_2 -> AVertexTerminal	B_4-> ABaseTerminal
C_3	B_2	B_5	u	Line	B_2-> ARandomTerminal	B_5 -> AVertexTerminal
C_4	B_5	B_3	u	Line	B_5 -> AVertexTerminal	B_3-> ARandomTerminal
C_5	B_3	B_6	m	Line	B_3-> ARandomTerminal	B_6-> AVertexTerminal
C_6	B_1	B_6	u	Line	B_1-> ARandomTerminal	B_6-> AVertexTerminal
C_7	B_1	B_11	N/A	Line	B_1-> ARandomTerminal	B_11-> ARandomTerminal
C_8	B_1	B_12	N/A	Line	B_1-> ARandomTerminal	B_12-> ARandomTerminal
C_9	B_1	B_13	N/A	Line	B_1-> ARandomTerminal	B_13-> ARandomTerminal
C_10	B_3	B_31	N/A	Line	B_3-> ARandomTerminal	B_31-> ARandomTerminal

Fortunately, since process flow diagrams and ER diagrams both belong to Block Diagram, we can simplify this process by modifying the existing process flow diagram evaluation ontology. Like the assumptions we made in Case Study 1, the expert first uses the drawing tool to choose representation shapes and to complete a diagram. Each representation has a class to model its properties. As described in Section 6.4, we have defined seven classes in ER diagram. Once the expert finishes the diagram, instances of *BLOCKs* and *CONNECTIONs* will be created automatically based on the assumption of our approach (Table 6.7 and Table 6.8).

Then an evaluation process will be created by consulting Figure 6.4 and Figure 6.3, after combing information from Table 6.7 and Table 6.8.

a) General property check. By consulting Figure 6.4, we notice that the ER diagram is in the association category with an additional property node "uniqueness." This indicates that the general property check in ER diagram includes *uniqueness*, in addition to the two general features, *connectivity* and *integrity*, applicable for all block diagrams. Therefore, compared with the existing process flow diagram evaluation ontology, the node *flow* should be replaced by *uniqueness*.

b) Block property check. In the ER modeling domain, new classes need to be constructed to capture types of block's representation by inheriting the abstract *REPRESENTATION* class. Therefore, the classes defined in process flow diagram cannot be reused. In an ER diagram, four classes are needed: *ENTITY, ATTRIBUTE, RELATION* and *ISA*, which have been discussed in Section 6.4. In the evaluation ontology, the four representation classes will be attached to the representation ovals under the block property check step. Then the evaluation for each representation block will be forwarded to the block property ontology for ER diagram (Figure 6.5).

c) Problem-specific check. Even though a process flow diagram is quite different from an ER diagram, our approach uses a similar inner representation to process the diagram and construct instances of *BLOCK* and *CONNECTION*. Therefore, only values in purple nodes in Figure 6.15 need to be modified by consulting Table 6.7 and Table 6.8. Categories of conceptual connections need to incorporate the current domain concepts (blue nodes on the right side in Figure 6.16), and the representation class can be related (green nodes on the left side in Figure 6.16).

Moreover, feedback from pedagogy module needs to get adjusted accordingly. The domain expert has to specify feedback to tackle each type of instruction for when an error is detected.

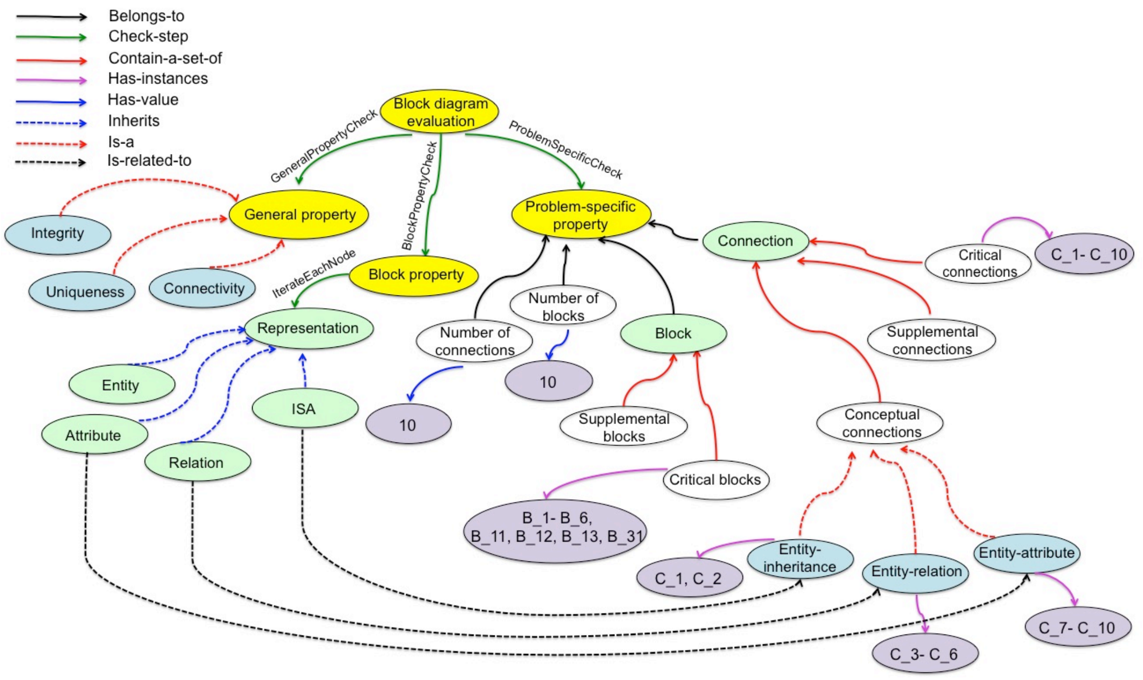


Figure 6.15. Evaluation ontology for a personnel ER modeling. Yellow nodes: 3-step evaluation process. Note: the general property ontology (Figure 6.4) is required to retrieve diagram's properties for general property check. The block property check for each representation class will be forwarded to a domain-dependent block property ontology (Figure 6.5). Green nodes: abstract classes that are domain-dependent. White nodes: data structures that contain information from a diagram. Purple nodes: information/instances retrieved from an expert diagram. Detailed information is shown in Tables 6.7 and 6.8. Blue nodes: general property or conceptual connection subtypes that belong to a given domain. Across-domain nodes: yellow and white; Domain-dependent nodes: blue and green; problem-dependent nodes: purple.

In Case Study 2, we applied our method to ER diagrams based on what we had created in Case Study 1 for process flow diagrams. The purpose of giving this example is to demonstrate how our method can be easily transferred to another domain. As we stated before, representation classes are domain-dependent. Therefore, classes from process flow diagram cannot be re-used. So the major effort of creating a new domain tutor design is to make corresponding representation classes. Then it is a simple routine to get instances of *BLOCKs* and *CONNECTIONs* from the diagram. The evaluation ontology retains a similar structure, except that conceptual connections need to be defined specifically for each area.

Case Study 3. Create an evaluation ontology to handle a new problem in ER modeling, given that an existing ontology for an ER diagram is available from Case Study 2.

Now the expert wants to add another ER diagram problem (Figure 6.16) to the existing ER diagram tutoring system. What changes does she need to make?

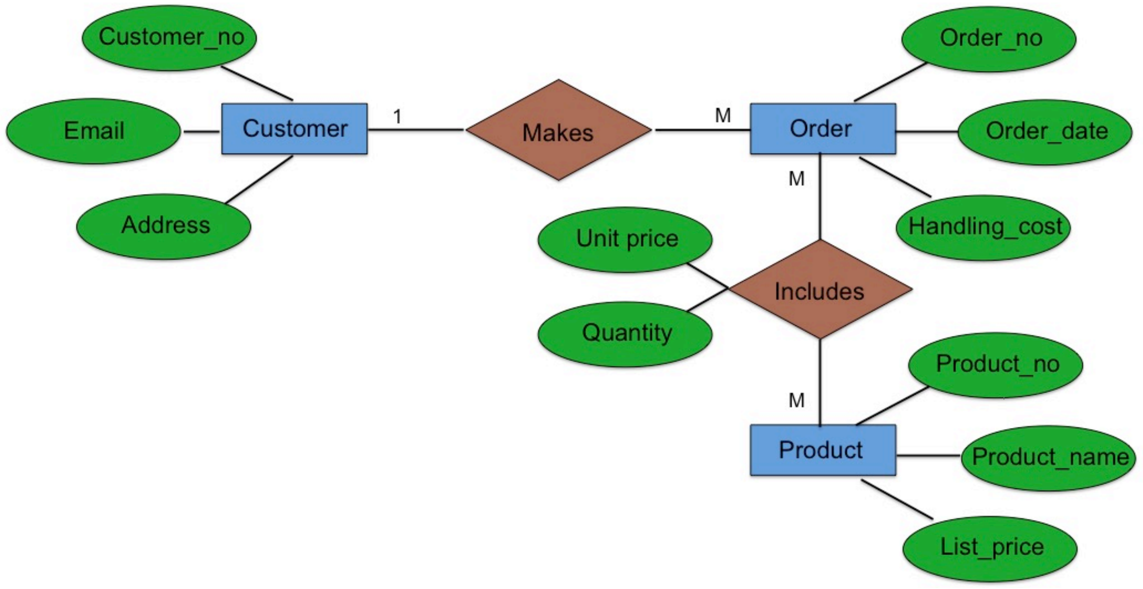


Figure 6.16. An online shopping database ER modeling diagram.

As the expert has an existing ER diagram tutoring system design with all the representation classes available, according to Figure 6.3, she doesn't need to make any changes in general-property check step and block-property check step. Instead, only problem-specific properties need to be modified (the purple nodes in Figure 6.16). As is shown in Figure 6.3, it contains seven components: *critical blocks, supplemental blocks, critical connections, supplemental connections, number of blocks, number of connections* and *conceptual connections*. Based on our earlier explanation, the system should identify the value of each component by either analyzing expert's diagram, or allow the expert to input the information manually. Therefore, to establish a knowledge representation of problem-specific information from the expert diagram, it can go over each block in the diagram and retrieve the text contents from it, i.e., construct class *CONTENT* and class *OPERATION* if applicable. In other words, instances of the *BLOCK* class can be built easily by parsing the expert's solution without extra effort to construct domain knowledge representation (Table 6.9 and Table 6.10). Notice this new problem doesn't have an instance of *ISA* class. Therefore, *ISA* is disabled in the representation class and in conceptual connection. *Entity-inheritance* is not necessary anymore (shown in grey node in Figure 6.17).

Still, no change is needed in the pedagogy module. Notice that five types of feedback in the pedagogy module are related to problem-specific check results. Misconception type is the only one that might be neglected if there is no conceptual connection existing in the expert solution. Once the information is captured, the ontology is ready to be implemented to tutor the next problem.

Table 6.9. Instances of *BLOCKs* in online shopping database ER modeling.

Block	**Content**->label	**Representation**	**Operation**
B_1	Customer	Entity	N/A
B_2	Makes	Relation	N/A
B_3	Order	Entity	N/A
B_4	Includes	Relation	N/A
B_5	Product	Entity	N/A
B_11	Customer-no	Attribute	N/A
B_12	Name	Attribute	N/A
B_13	Email	Attribute	N/A
B_31	Order-no	Attribute	N/A
B_32	Order-date	Attribute	N/A
B_33	Handling-cost	Attribute	N/A
B_41	Unit-price	Attribute	N/A
B_42	Quantity	Attribute	N/A
B_51	Product-no	Attribute	N/A
B_52	Product-name	Attribute	N/A
B_53	List-price	Attribute	N/A

Table 6.10. Instances of *CONNECTIONs* in online shopping database ER modeling.

Connection	Node_1	Node_2	Label	Connecting type	Point_1	Point_2
C_1	B_1	B_2	1	Line	B_1-> ARandomTerminal	B_2 -> AVertexTerminal
C_2	B_2	B_3	M	Line	B_2 -> AVertexTerminal	B_3-> ARandomTerminal
C_3	B_3	B_4	M	Line	B_3-> ARandomTerminal	B_4 -> AVertexTerminal
C_4	B_4	B_5	M	Line	B_4 -> AVertexTerminal	B_5-> ARandomTerminal
C_5	B_1	B_11	N/A	Line	B_1-> ARandomTerminal	B_11-> ARandomTerminal
C_6	B_1	B_12	N/A	Line	B_1-> ARandomTerminal	B_12-> ARandomTerminal
C_7	B_1	B_13	N/A	Line	B_1-> ARandomTerminal	B_13-> ARandomTerminal
C_8	B_3	B_31	N/A	Line	B_3-> ARandomTerminal	B_31-> ARandomTerminal
C_9	B_3	B_32	N/A	Line	B_3-> ARandomTerminal	B_32-> ARandomTerminal
C_10	B_3	B_33	N/A	Line	B_3-> ARandomTerminal	B_33-> ARandomTerminal
C_11	B_4	B_41	N/A	Line	B_4-> ARandomTerminal	B_41-> ARandomTerminal
C_12	B_4	B_42	N/A	Line	B_4-> ARandomTerminal	B_42-> ARandomTerminal
C_13	B_5	B_51	N/A	Line	B_5-> ARandomTerminal	B_51-> ARandomTerminal
C_14	B_5	B_52	N/A	Line	B_5-> ARandomTerminal	B_52-> ARandomTerminal
C_15	B_5	B_53	N/A	Line	B_5-> ARandomTerminal	B_53-> ARandomTerminal

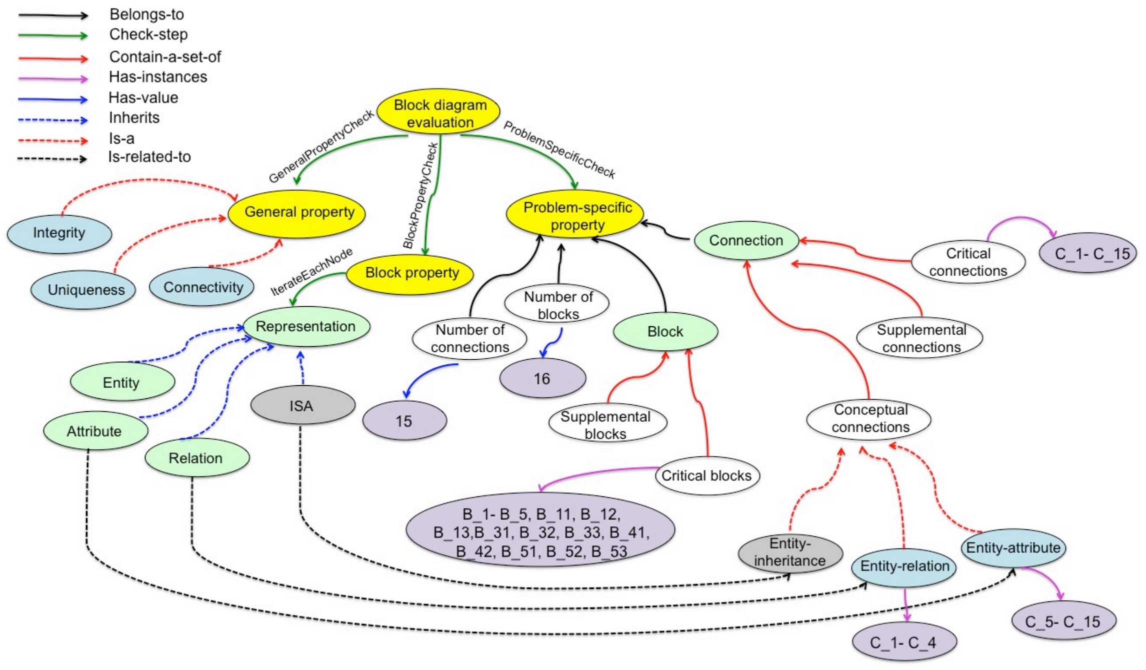


Figure 6.17. Evaluation ontology for an online shopping database ER modeling. .Yellow nodes: 3-step evaluation process. Note: the general property ontology (Figure 6.4) is required to retrieve diagram's properties for general property check. The block property check for each representation class will be forwarded to a domain-dependent block property ontology (Figure 6.5). Green nodes: abstract classes that are domain-dependent. White nodes: data structures that contain information from a diagram. Purple nodes: information/instances retrieved from an expert diagram. Detailed information is shown in Tables 6.9 and 6.10. Blue nodes: general property or conceptual connection subtypes that belong to a given domain. Gray nodes: domain-dependent nodes that are not used in this problem. Across-domain nodes: yellow and white; Domain-dependent nodes: blue, green and gray; problem-dependent nodes: purple.

In summary, if an expert wants to create a new problem tutor design in the same domain, she doesn't need any modification of the pedagogy module. In evaluation steps, the general property and block property checks will remain the same, as they are domain-dependent. She only needs to update the problem-specific property in order to handle the new problem, i.e., update instances of the seven white nodes in Figure 6.17. This case study shows that our method has a low entry point to generate new problems within a domain. This is the situation that most tutor design occurs. A domain expert who has a specific need in one type of diagram always has the need to create several problems to address the principles.

In this section, we offered three case studies to illustrate how our method can be generalized. It can be helpful in creating a diagram tutor design from scratch. Also, it can be easily transferred to another type of diagram, and support the creation of a new problem within the same domain. The evaluation ontology can be used as the default knowledge representation in a Diagram-based ITSs authoring tool. The three-step evaluation (yellow nodes and white nodes) can be implemented as the default process that only requires experts to customize based on their needs. The general property ontology (Figure 6.4) can serve as basis for representing knowledge in the authoring tool. When authoring the general property check, domain experts can retrieve the general properties associated with their target diagram by choosing from a list of existing Block Diagrams. If their interested diagrams are not on the list, they can access the general property ontology and find their diagram's subcategory. Properties can be defined for this new Block Diagram domain. Thus, the general property ontology can be expanded by domain experts. In addition to the yellow nodes and the white nodes, the abstract class nodes (green ovals in Figure 6.3) will be extended by adding the domain-dependent representation nodes (green ovals in Figure 6.18) to facilitate the block property check. These domain-dependent representation nodes can be either articulated by experts' input, or automatically entered by the system if the block representation reveals their identities. Finally, after the experts upload their correct diagram solution, all the purple nodes can be created when the system parses the diagram.

The benefit of this ontology becomes more clear when one imagines attempting to create an evaluation process from scratch, or trying to create an evaluation process for one domain based on the evaluation process of another domain. If there is no ontology, or the ontology is empty, careful planning and thinking is required. If the components of Figure 6.3 are all present, any appropriate components can be re-used in the new domain.

6.7. Discussion and Conclusion

In this chapter, we chose Block Diagram as our target interest, which contains geometric shapes and connections to demonstrate relationships, flow, pattern, etc. A general property ontology is provided which categories Block Diagram into five subtypes, where each subtype is given properties based on its intrinsic relations.

To address the research question asked at the beginning of this chapter, we used abstract classes to model the geometric representations in Block Diagram, such as *BLOCK, REPRESENTATION, CONTENT, OPERATION* and *CONNECTION*. It provided a modular way to allow the evaluation system to parse the diagram and build internal knowledge representation. The diagram evaluation ontology was built on top of this representation, which included three steps: 1) general property check, which evaluates if the given diagram satisfies the properties from the general property ontology; 2) block property check, which detects if each block is appropriately represented and if the properties associated with each block's representation include legal number of connections and permitted connecting types (domain-dependent); and 3) problem-specific check, which checks if the given diagram has similar information as the expert's. In particular, this ontology constructs an internal representation of the knowledge from the diagram by using the classes we defined before, and compares it with the expert's knowledge, which is problem-dependent. Conceptual knowledge can be also included in terms of conceptual connections between blocks. Furthermore, pedagogical instruction will be given based on the outputs of the evaluation process. In particular, one type of instruction will be provided to the general property and block property check, whereas five categories of instruction can be used by the problem-specific check, such as insertion, omission, replacement, reversion of direction and misconception. Thus, we answered our question by constructing abstract classes as knowledge representation for Block Diagram, and developing the general property ontology and evaluation/pedagogy ontology to set as designs for a future Diagram-based ITS authoring tool.

To demonstrate with examples, three applications in process flow diagram, entity-relationship diagram and circuit diagram were discussed. Block properties and evaluation ontologies were provided. Also, a more general type that doesn't have block properties, concept map, was included. From those examples in different domains, we showed our general ontology and evaluation/pedagogy ontology were indeed a solution to ease the expert's process and help to describe domain knowledge in authoring Diagram-based ITSs.

Then we gave three case studies to validate our class representation and evaluation/pedagogy ontologies. 1) The process involved in creating a new tutor design from scratch using our ontologies; 2) the process of creating a tutor design in another domain based on the existing ontology and the changes needed (big transfer); 3) the process needed in adding a new problem within the same domain (small transfer). As we mentioned before, it is hard to construct an ontology when starting from scratch. However, when the backbone of the evaluation ontology is constructed, it supports information reuse and edits. The ontological approach saves domain experts time and effort if a new series of problems are required. In the future, ontology generation could be enabled by an input system. I.e., by filling domain knowledge rules associated with each block, the system could generate an ontology that is readable by the computer. For example, the block property ontology for ER diagrams could be generated by asking experts a couple of questions about each block, such as the maximum number of connections, legal connecting types, etc.

Our method brings an important methodology for the design of future authoring tools for Diagram-based ITSs. First, the general property ontology provides guidelines on diagramming based on the intrinsic properties that each type of diagram category possesses. This property merely depends on the domain. We believe the existing diagrams in STEM which bear the definition of Block Diagrams belong to one of the five subcategories we defined in this chapter. In that sense, ideally, our method can be applied to any Block Diagram in STEM.

Secondly, the block property ontology carries domain-dependent knowledge, which supports knowledge reuse by domain experts. Once the representation class is created, it can be reused across problems within the domain. New types of block can be easily added to the existing representations, if the expert wants to extend the domain knowledge. A future authoring tool to construct the block property ontology will be beneficial. Furthermore, the block property ontology can be used not only to diagnose student's drawing, but also to serve as expert knowledge to validate an expert's solution. I.e., when the expert has errors in the solution, the block property ontology is able to detect them and further give hints to correct them. For instance, in an ER diagram, the expert solution has two entities linked together by accident. In the block property ontology, it says an *ENTITY* can only be connected to a *RELATION* or *ATTRIBUTE* or *ISA*. By consulting the block property ontology, it is easy to find this violates the domain knowledge. On the other hand, the block property ontology can be revised by the expert if its current representation is not sufficient or inappropriate. There are cases when it is created but used by different experts. So allowing the block property ontology to be updated by the current expert would be a convenient solution.

Moreover, if experts have different evaluation approaches, our method could help them to align their ideas. For example, in the domain of process flow diagrams, Expert A might prefer to first check the existence of the blocks, and then check if there are flows among the blocks. Expert B, on the other hand, who uses the same ideas as our method, wants to check if the diagram is a valid process flow diagram, e.g., whether arrows are used to link blocks. Both evaluation processes are appropriate as they focus on different priorities. Thus, a student's diagram would receive different instructions based on their different evaluation processes. However, our method will provide an insight to Expert A that checking the validity of a process flow diagram should possibly receive higher priority than his or her first check on block existence, as it applies to any process flow diagrams.

However, in the current stage, our method cannot optimize an expert's solution, as we don't have a sophisticated model to do reasoning. The problem-specific check only depends on an expert solution, i.e., the number of blocks, number of connections, instances of blocks and connections. In some cases, the expert's solution might not be the optimum solution in terms of using the fewest blocks and connections to represent the required contents. Also, when there might be more than one solution to a problem, our approach will expect the expert to provide all alternative solutions and check if student's solution satisfies anyone of them. Further improvements are needed to solve these situations. For example, like constraint-based modeling, we could add an equivalence property to our block property ontology. The equivalence property would specify the possible equivalent representations, e.g., an item can be replaced by another item within the group. For instance, in an ER diagram, a *WEAK ENTITY* can be modeled by an *ATTRIBUTE*.

To summarize, by using abstract classes to represent diagram knowledge and ontologies to help evaluation process and pedagogical feedback generation, we answered our research questions for this chapter: "Is there a general way to represent a diagram's domain knowledge that is applicable to various types of diagrams, to support the authoring of a Diagram-based ITS? Can we also create evaluation procedures and tutoring instructions on top of it?" Case studies in three situations validated that the ontologies are applicable to different areas across different types of diagrams. We noted that our approach will be a good solution to support the design of future Diagram-based ITSs. It not only makes the knowledge representations easier to construct, but also it builds a modular structure to evaluate a diagram, which can be generalized to various types of diagrams in the family of Block Diagram.

REFERENCES

Aroyo, L., & Dicheva, D. (2004). The new challenges for e-learning: The Educational Semantic Web. *Educational Technology and Society*, *7*(4), 59–69.

Borst, W. (1997). *Construction of engineering ontologies for knowledge sharing and reuse*. Centre for Telematics and Information Technology.

Chen, W., Hayashi, Y., Jin, L., Ikeda, M., & Riichiro Mizoguchi. (1998). An Ontology-based Intelligent Authoring Tool. *Proc. of the Sixth International Conference on Computers in Education*, 41–49.

Gomez-Perez, A., & Corcho, O. (2002). Ontology languages for the semantic web. *IEEE Intell Syst*, *17*(1), 54–60.

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowl Acquis*, *5*(2), 199–220.

Grundspenkis, J., & Anohina-Naumeca, A. (2015). *Fundamentals of Artificial Intelligence: knowledge representation and networked schemes*. Lecture notes. Retrieved March 3, 2015, from http://stpk.cs.rtu.lv/sites/all/files/stpk/lecture_7.pdf

Guarino, N., & Welty, C. (2002). Evaluating Ontological Decisions with OntoClean. *Commun. ACM*, *45*(2), 61–65. doi:10.1145/503124.503150

Hager, P. J., Scheiber, H. J., & Corbin, N. C. (1997). *Designing & Delivering: Scientific, Technical, and Managerial Presentations*. New York ; Toronto : J. Wiley.

Ikeda, M., Seta, K., & Mizoguchi, R. (1997). Task ontology makes it easier lb use authoring tools. *IJCAI International Joint Conference on Artificial Intelligence*, *1*, 342–347.

Khoo, C. S. G., & Na, J.-C. (2006). Semantic relations in information science. *Annual Review of Information Science and Technology*, *40*(1), 157–228. doi:10.1002/aris.1440400112

Larkin, J. H., & Simon, H. A. (1987). Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, *11*(1), 65–100. doi:10.1111/j.1551-6708.1987.tb00863.x

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In S. Murray, T., Blessing, S., Ainsworth (Ed.), *Authoring tools for advanced learning environments*. Kluwer Academic Publishers, Dordrecht.

Nakatsu, R. T. (2010). *Diagrammatic reasoning in AI*. John Wiley & Sons, Inc.

Noy, N. F., & McGuinness, D. (2001). Ontology development 101: A guide to creating your first ontology. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, 1–25. doi:10.1016/j.artmed.2004.01.014

Py, D., Alonso, M., Auxepaules, L., & Lemeunier, T. (2008). Design of Pedagogical Feedbacks in a Learning Environment for Object-Oriented Modeling. In

Proceedings of Educators Symposium at the 11th IEEE International Conference MODELS (pp. 39–50).

Smiciklas, M. (2012). *The Power of Infographics: Using Pictures to Communicate and Connect with Your Audience.* Pearson education, Inc.

Studer, R., Benjamins, V., & Fensel, D. (1998). Knowledge engineering: principles and methods. *Data Knowledge Engineering*, *25*(1-2), 161–197.

Systems Modeling Language. (2015). Retrieved June 29, 2015 from http://www.omgsysml.org

Zouaq, A., & Nkambou, R. (2009). Enhancing Learning Objects with an Ontology-Based Memory. *IEEE Transactions on Knowledge and Data Engineering*, *21*(6), 881–893.

Zouaq, A., & Nkambou, R. (2010). A survey of domain ontology engineering: methods and tools. In R. Nkambou, J. Bourdeau, & R. Mizoguchi (Eds.), *Advances in Intelligent Tutoring Systems*. Springer Berlin Heidelberg.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1. Conclusion

In this chapter, we will give a brief summary of our work, and discuss each piece of the work to answer our research questions in Chapter 1.

Generally speaking, our work contains two contributions. First, we developed two Diagram-based Intelligent Tutoring Systems (ITSs) to understand their usage. One is the StaticsTutor for free-body diagrams, which helps undergraduates to focus on the conceptual understandings but avoid numeric values or equations during the problem framing stage. The other one is the Thermo Cycle Tutor for thermodynamics cycle T-v diagrams. It implemented decision-based pedagogical strategies from a domain expert in an undergraduate thermodynamics course. Initial investigations of several groups of students across four semesters and a few instructors have shown that StaticsTutor can effectively identify students' misconceptions, whereas Thermo Cycle Tutor can increase students' learning gain very quickly, as shown by comparing their pretest and posttest scores.

These two ITSs answered our first question: what can be learned from creating tutors with different underlying representations?

From the implementation perspective, we learned that software classes can be used to represent geometric shapes in a diagram. Even though a T-v diagram has a different structure than a free-body diagram, we used a similar method to parse the diagram and create inner class representation. T-v diagrams, which are more complicated, need higher-level classes to reveal the conceptual level representation. For instance, a pressure line in a T-v diagram constitutes one horizontal line and two positive-sloped lines. So a *PRESSURE LINE* class was constructed which contains three instances of the class *LINE*. Methods within each class are also developed, to deal with properties or operations to instances of other classes or to itself. For example, *above (a line)* in class *POINT*, is a Boolean method, which returns true if an instance of class *POINT* is above an instance of class *LINE*. Therefore, we found that class representation is a general solution to knowledge representation on different types of diagrams.

From a pedagogical perspective, since research in ITS is not only to diagnose a student's solution, but also to improve a student's learning by giving tutoring instructions, we proposed some pedagogical guidelines in Chapter 5 that we learned from the two tutors. Tutoring a student's diagram can be difficult. One big issue is that conceptual knowledge might not be properly presented through diagrams. I.e., students understand the concept, but due to lack of procedural expertise or carelessness, they cannot convey it correctly in the diagram. Thus we suggested some methods, such as allowing different tolerances at different levels of evaluations, using Q&A to reduce ambiguity, determining if conceptual knowledge can be applied by procedure expertise in the current drawing, and determining if the new evidence violates the existing understanding of the student.

Second, to answer Research Questions 2 and 3, which are about the generality of how the current approach can be applied to other domains, we developed the following.

1) We chose a target. As we noticed that many diagrams contain geometric objects and use lines/arrows to connect them, such as process flow diagram, entity-relationship

diagram, circuit diagram, concept map, etc., we abstracted them as Block Diagrams. Based on Nakatsu (2010), we categorized Block Diagrams into five subtypes, on which a general property ontology was developed. Property nodes and instances nodes were added to make the ontology workable for Diagram-based ITSs.

2) We modeled a Block Diagram through classes. Knowledge representation on Block Diagrams was modeled by abstract classes, such as *BLOCK, REPRESENTATION, CONTENT, OPERATION* and *CONNECTION*. The *REPRESENTATION* class needs to be extended to actually represent objects in a domain. One example is the class *ENTITY* in ER diagram. It is inherited from *REPRESENTATION* class. Once a representation class is defined, it can be reused across problems. The class modeling allows the evaluation system to parse the diagram and build internal knowledge representation. Also, a block property ontology is defined to deal with the appropriate representation of each block, such as the number of connections it can have and the type of blocks it is allowed to link or must be linked.

3) We developed an evaluation/pedagogy ontology on top of the class representation and two property ontologies. In general, the evaluation/pedagogy ontology contains three steps: 1) general property check, which checks if the given diagram satisfies the properties from the general property ontology; 2) block property check, which detects if each block is appropriately represented; and 3) problem-specific check, which checks if the given diagram contains similar information to what could be in the expert's solution. In particular, the general property check and block property check are based on the information from the two ontologies in items 1 and 2. Conceptual knowledge can be also implied through conceptual connections between blocks, which

can be either user-defined or system-automated. Seven pedagogical instruction nodes were added to the evaluation/pedagogy ontology to handle the outputs of the evaluation process. We also applied our approach to three domains: process flow diagrams, ER diagrams and circuit diagrams, to further demonstrate it is workable across different domain areas. To validate our class representation, property ontology and evaluation/pedagogy ontology, three case studies were provided. The case studies demonstrated the process when a new tutor design is created from scratch, the process when it moves to a new domain and what changes are needed, and the process of adding a new problem in an existing domain.

To summarize, we believe our work will contribute to the development of the future Diagram-based ITSs authoring tools. In our vision, the future authoring system would implement the evaluation/pedagogy ontology as a default structure. The general property ontology could be loaded in the system too, allowing experts to choose their domain of interest. Once it is chosen, the relevant objects in a drawing palette would be presented to the user. Domain experts would care most about the creation of the block property ontology and classes. So an effective UI to make these steps simpler is desired. Also, it is expected to support class editing and reuse by existing experts, e.g., adding or removing a property from an existing class created by other experts. Furthermore, the tool should support authoring an expert solution and automatic conversion of geometric objects to class representations. By this means, an authoring system would be able to convert a drawing to the system's inner representation with the least efforts by domain experts, and the evaluation process and pedagogical instructions would be constructed seamlessly to support diagnosis and tutoring.

7.2. Future Work

This section describes how the process we followed might be extended to other diagrams beyond Block Diagrams. The Thermo Cycle Tutor helped undergraduates with T-v diagrams in a thermodynamics course. Unlike a Block Diagram, it contained two axes (temperature and volume) where the location of states (points) was considered. This type of diagram is an example of a large category of diagrams in STEM. Figure 7.1 shows one example of P-v diagram in a diesel cycle. Even though the diagram contains four elements and there are some connections among them, it is different from a Block Diagram. The location of the elements is of interest and plays a crucial role in conveying messages from the diagram. Instead, Block Diagram doesn't model the location of each element/block.

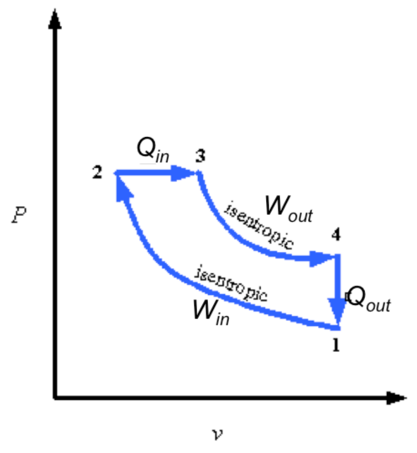


Figure 7.1. P-v diagram for a diesel cycle.

We call this type of diagram a Quantitative Diagram. It includes an x and y axis to demonstrate the relations of two variables in two-dimensional space. Data points are presented to show trends, changes or separations. In addition, it might contain other objects other than points to facilitate the demonstration.

Like a Block Diagram, we can use classes to model internal knowledge within the Quantitative Diagram. However it needs more complexity. Some classes in Quantitative Diagram are listed below:

1) Axis. It is a part of the Cartesian coordinate system. Several attributes can be included, such as the axis name, unit, maximum/minimum value it possesses.

2) Point. It is the atomic type in the drawing. Its x and y values, label or descriptions can be modeled as attributes.

3) Connection. It models the linkage or association between points. This class is similar to the *CONNECTION* class in Block Diagram, which might contain the two connecting nodes, connecting type and labels along with the connection.

4) Special object. This type is unique in Quantitative Diagrams. As we discussed before, a T-v diagram uses a vapor dome to separate the 2-d space to five regions. Locations of each point regarding to the vapor dome is important. In this case, vapor dome can be modeled as a special object. Note, it is not usual to have special object in Quantitative Diagrams, and the shape and properties of special object vary from domain to domain. Therefore, this type needs domain experts to define explicitly based on their purpose.

Moreover, the connecting type needs more sophisticated consideration. In Block Diagrams, only two types are modeled: *LINE* and *ARROW*. Since we only care about if the two blocks are connected, and if the relationship such as flow, cause/effect or inheritance between them is correct, it is sufficient to model a connecting type with a direction (arrow) or non-direction (line). However, in a Quantitative Diagram, the connecting type becomes more sensitive. Besides line and arrow, there could be more types. As shown in Figure 7.1, a concave upward curve is used to model an isentropic transition from state 1 to 2 during the compression stage. Also, an arrow can be added to the curve to show the flow. For instance, the curve from 1 to 2 has different directions than that of 3 to 4.

Based on domain requirements and potential applications, the Quantitative Diagram can have many varieties. Some of them focus on the points' relative spatial relation and how they are connected (by line or arrow or curve). An example is the P-v diagram. Some of them only consider the points' absolute location in respect to the two axes whereas the connections are less important. The velocity-time diagram, for example, contains velocity at each time moment, which will be of interest. Some of them have additional concerns on the extreme points, e.g., a local minimum or maximum. For instance, an enzyme reaction rate diagram, in which the rate increases as the substrate's concentration increases until it reaches a constant rate. Therefore, the point that represents the constant reaction rate will be an extreme point and is one of the big concerns in the diagram diagnosis. Thus, we would suggest categorize Quantitative Diagram into several subtypes and abstract the properties/requirements of each type. Like Block Diagram, a general property ontology can be used by diagram evaluation.

To evaluate Quantitative Diagrams, we can still apply a similar method to what we used in Block Diagrams: an evaluation ontology. However, as there isn't a block property within each point, the block property check will be removed. So, two steps can be involved: a general property check and problem-specific check. The general property check can be based on a general property ontology as we just mentioned. The problem-specific check will apply the class representations to retrieve *point-wise* and *connection-*

wise information from the diagram, e.g., x-axis, y-axis, number of points, number of connections, instances of points and instances of connections. As in Block Diagrams, points/connections can be labeled as critical or supplemental. However, it is hard to automate this process, as there isn't a shape associated with a point. Conceptual knowledge can also be embedded, e.g., an extreme point in some diagrams reveals how students perceive the scientific process based on their domain knowledge. Pedagogy instructions can be extended from what we discussed in Block Diagram. More careful design is needed to address a point's wrong spatial location with respect to the axes, or relative spatial location among points.

To conclude, the methods we used to model Block Diagrams can be used to model Quantitative Diagrams, which capture spatial relations among points in the Cartesian coordinate system. Like a Block Diagram, it can be categorized into a few subtypes according to the requirements or features in the diagram, on which general property check can be applied. Since it is more sensitive than a Block Diagram to the objects' geometric location and the way of connecting them, additional or more sophisticated classes, such as *AXIS* and *POINT* are needed to fulfill the purposes. On the other hand, object's representation is not considered in Quantitative Diagrams, as all the objects are abstracted as points. Thus, only two evaluation steps are needed in Quantitative Diagram tutoring systems: general property check and problem-specific check.

This example is just one application of potential future work of this research. The example shows that our approach may be a solution to other types of diagram with some modifications. We look forward to the application of these methods to a wide variety of complex student knowledge representations, esp. other categories of diagrams.

REFERENCES

Nakatsu, R. T. (2010). *Diagrammatic reasoning in AI*. John Wiley & Sons, Inc.