Revisiting Ill-Definedness and the Consequences for ITSs

Antonija MITROVIC and Amali WEERASINGHE
Intelligent Computer Tutoring Group
University of Canterbury, Private Bag 4800, Christchurch, New Zealand
tanja.mitrovic@canterbury.ac.nz, amali.weerasinghe@pg.canterbury.ac.nz

Abstract: ITSs for ill-defined domains have attracted a lot of attention recently, which is well-deserved, as such ITSs are hard to develop. The first step towards such ITSs is reaching a wide agreement about the terminology used in the area. In this paper, we discuss the two important dimensions of ill-definedness: the domain and the instructional task. By the domain we assume declarative domain knowledge, or the domain theory, while the instructional task is the task the student is learning, in terms of problem-solving skills. It is possible to have a well-defined domain and still have ill-defined instructional tasks in the same domain. We look deeper at the features of ill-defined tasks, which all contribute to their ill/well defined nature. The paper discusses model-tracing and constraint-based modeling, in terms of their suitability for ill-defined tasks and domains. We show that constraint-based modeling can be used in both well- and ill-defined domains, and illustrate our conclusion using several instructional tasks.

Introduction

Recently there has been a lot of attention on supporting learning in ill-defined domains (aka ill-structured domains), as evidenced by the three workshops held in 2006-8 [1-3]. This attention is welcome, as ITSs for such domains are rare, and usually much more demanding than the typical ITSs for well-defined domains. However, there has been little agreement about the terminology used and the underlying definitions between researchers working in this area, even among those who presented their work at the workshops. Most researchers equate ill-defined domains with ill-defined tasks [1-3]. In this paper, we argue that it is important to make the distinction between instructional domains and tasks. We start by discussing instructional domains and tasks as two important dimensions in the light of ITSs. Section 2 presents a deeper discussion of instructional tasks, focusing on various factors which influence the nature of the task. We then turn to student modeling approaches which are appropriate for various instructional situations, and then show how ITSs can deal with ill-definedness.

1. A deeper look at ill-definedness: the two dimensions

An instructional domain is an area of study, such as mathematics or philosophy. In order to learn a particular instructional domain, the student needs to learn the relevant declarative knowledge (i.e. the domain theory), and in many domains also needs to

acquire problem-solving skills. ITSs are almost exclusively problem-solving environments, based on the assumption that students have learnt the declarative knowledge from direct instruction (lectures, books and/or peers) and only need to practice their problem-solving skills [4, 5]. Most ITSs provide lots of problem-solving opportunities and only occasionally give direct instruction, in the form of examples or definitions of the concepts used as in [6]. There are also ITSs that provide instructional material in addition to problem-solving support, such as ELM-ART [7], but in this paper we will focus on problem-solving as the main instructional activity.

It is important to make a clear distinction between those two types of learning (acquiring declarative knowledge versus problem-solving skills) for the discussion of ill-definedness. We found a lot of confusion in published papers when discussing ill-definedness. Most researchers equate ill-definedness with the underlying domain theory, and provide examples of ill-defined domains, such as essay writing. Commonly used examples of well-defined domains are mathematics and physics. However, there seems to be no differentiation between the characteristics of domains versus tasks.

We propose that two orthogonal dimensions need to be considered when discussing ill-definedness: the domain, and the task. Starting from our first dimension, domains vary in terms of their underlying domain theories. There are many domains covered by ITSs that are completely well-defined, such as many areas of mathematics, physics and chemistry. Instructional tasks that they teach are also well-defined: for example, adding fractions, solving equations for unknowns, or balancing chemical equations. The student is taught the theory, as well as the procedure (i.e. the algorithm) to use to solve problems. Such domains are in the WDWT quadrant in Figure 1.

However, if the domain is well-defined, that does not necessarily mean that instructional tasks in that domain will also be well-defined. As an illustration, let us focus on the domain of database design [8]. Conceptual database design is a task of converting the database requirements into a high-level description of the database, most often expressed in terms of the Entity-Relationship (ER) data model [8]. On the other hand, logical database design is a process of converting the ER diagram into a relational schema, thus requiring an understanding of the relational data model. Both the ER and relational data models are well-defined: they consist of a small number of components with well-defined syntax and semantics. Although the ER model itself is well-defined, the task of developing an ER schema for a particular database (i.e. conceptual database design) itself is ill-defined: the initial state (i.e. the set of requirements) is usually underspecified and ambiguous, there is no algorithm to use to come up with the solution, and finally the goal state is also underspecified, as there is no simple way of evaluating the solution for correctness. Therefore, conceptual database design belongs to the WDIT quadrant in Figure 1. Logical database design, however, is well-defined, as there is a simple deterministic algorithm which guarantees good solutions (shown in the WDWT quadrant in Figure 1). Other examples for the WDIT quadrant include programming and writing SQL queries: although the relevant languages are well-defined, the task of converting the problem statement into a program is ill-defined.

Many domains are ill-defined, such as essay writing. In that case, the declarative knowledge is incomplete: it specifies how to structure the essay, how to present arguments, and also defines writing styles. The domain theory in this case is ill-defined, as is the task itself (writing the essay), as illustrated in the IDIT quadrant in Figure 1. The two dimensions are continuous; there is a spectrum arranging domains from ill- to well-defined ones, as well as another spectrum for instructional tasks. There

are some dependencies between them, as ill-defined domains usually involve ill-defined tasks, but the contrary is not necessarily so. Note that there are no examples for the IDWT quadrant: here the domain theory is ill-defined, but the task is well-defined. We believe this combination is not possible, and do not consider it further.

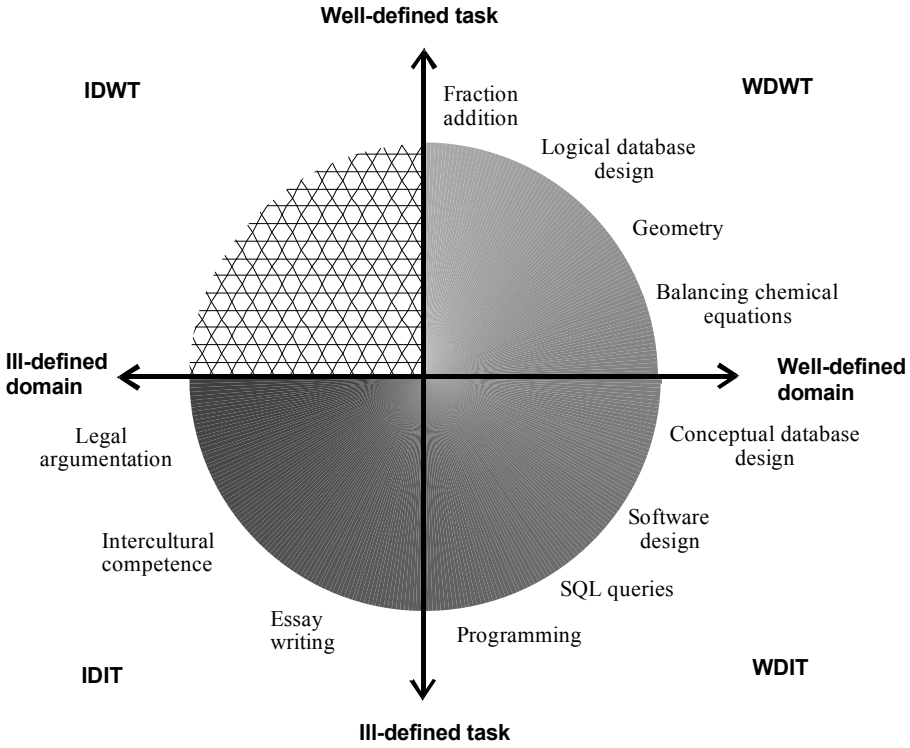

Figure 1. The space of instructional domains and tasks

2. Classifying instructional tasks: important characteristics

When discussing the definedness of decision-making tasks, there are four important factors to consider [9, 10]: start state, goal state, and the transformations (i.e. the problem-solving procedure), as well as the decision maker's familiarity with each of the factors. Decision making is similar to problem solving, and for that reason we adopt those factors. In addition, we add another one: the existence of a correct solution.

The initial state is presented to the student in the form of a problem statement. Instructional tasks taught to younger students most often have well-specified problems statements – e.g. simple arithmetic tasks, equation solving and other tasks in science. Problem statements for more challenging tasks can be less specified: in a typical university-level mechanics problem, the text of the problem does not specify all the forces acting on a given body. In conceptual database design or software design, the student is given a set of requirements, which is often incomplete and/or ambiguous. To deal with such problems, the student needs to use not only declarative domain knowledge they learnt previously, but also his/her world knowledge in order to

eliminate ambiguities and (when necessary) add missing information. Therefore, in order to deal will ill-defined problem statements, the student has to process the given information in order to complete the specification (and therefore turn the problem statement into a well-defined one).

Goal states can also be well- or ill-defined. In easy tasks, the student is clear about the form of the final solution. For example, the student had learnt that there were two solutions for a quadratic equation before attempting to solve any equations. When adding two fractions, the student knows that the solution should be (in the general case) another fraction. Additionally, the student can easily check whether the solution is correct or not. However, in design tasks, there is little information about the goal state. The goal state in such tasks is defined in a very abstract way; for example, in database design the goal state is defined as an ER diagram that is syntactically correct and matches the given requirements. Therefore, there is no simple test to use to check for correctness; the student can only apply the declarative knowledge he/she possesses in order to evaluate the solution produced. Another important issue is whether there is a stopping criterion – how can the student tell whether he/she is finished solving the problem? A well-defined goal possesses a stopping criterion which is easy to apply, while the ill-defined ones do not, and the student is again left to apply the constraints from the declarative knowledge in order to evaluate the solution.

Transformations or the problem solving procedure is another important factor. Some instructional tasks have a well-defined algorithm to apply to the initial state to derive the goal state. We have previously mentioned several tasks from mathematics, physics or chemistry with well-defined algorithms. In such situations, the student task is (relatively) easy: the student needs to memorize the algorithm and apply it correctly. Other similar examples involve some engineering tasks involving calculations. However, there is a very important subclass of tasks that deal with design. Design is in general ill-defined, as there are no algorithms to use to transform the initial state into the goal state. In addition, the start state is underspecified, and the goal state defined in terms of highly abstract features. Design tasks typically involve huge domain expertise, and large, highly structured solutions [11]. Typical examples of design tasks include architecture, software design, mechanical engineering and music composition. In such tasks there may be heuristic rules that can guide the student, but in general the student needs to apply the constraints from the domain theory. The other possible mechanism is analogy: the student can compare the current problem to those previously solved and perform analogical reasoning to deal with the complexity.

Finally, some researchers believe that ill-defined tasks are those that have no correct solution, but rather a family of solutions which can all be deemed correct. This is true of the extreme cases, such as essay writing: there might be any number of very good essays on a specified topic. In design tasks, there are often several (or even many) solutions that are all equally good. However, in a teaching situation, the teacher often has a good pedagogical reason for preferring one particular solution over the others. For example, in SQL there are often several correct queries for the same problem, differing from each other in the constructs used (please note that there is a lot of redundancy in SQL and, therefore, multiple ways of satisfying the same requirements). Even in such a task, the teacher may prefer one of those solutions among others; for example, the teacher may want to illustrate the use of a particular predicate. Therefore, it is still possible to nominate one “ideal” solution without compromising the quality of the whole ITS, as long as the ITS is capable of identifying other alternative solutions students may come up with as correct.

Table 1 presents the two dimensions and the factors of instructional tasks, and presents a few examples, categorized with respect to the factors discussed.

Table 1. Some examples of instructional tasks and their domains

Instructional task	Domain	Problem specification (initial state)	Goal specification (goal state)	Problem-solving procedure	Correct solution
Fraction addition	Well-defined	Well-defined	Well-defined	Well-defined	Only one
Balancing chemical equations	Well-defined	Well-defined	Well-defined	Well-defined	Only one
SQL queries	Well-defined	Ambiguous/ incomplete	Abstract	None	Multiple
Software design	Well-defined	Ambiguous/ incomplete	Abstract	None	Multiple
Essay writing	Ill-defined	Abstract	Abstract	None	Multiple
Legal argumentation	Ill-defined	Abstract	Abstract	None	Multiple
Intercultural competence	Ill-defined	Abstract	Abstract	None	Multiple

3. Student modeling approaches and ill-definedness

Another important issue is how the ITS models the student. Model tracing [4] is the most widely used student modeling approach currently. Since it tracks student's progress by generating solutions step-by-step, this approach is suited to well-defined tasks. Developing model-tracing tutors for ill-structured tasks is much harder, as it is difficult to come up with problem solvers for such tasks [1-3, 12]. However, constraint-based tutors do not suffer from such difficulties. Within the Intelligent Computer Tutoring Group (ICTG), we have developed constraint-based tutors for many tasks, both well- and ill-defined. Examples of our tutors teaching well-defined tasks range from fraction addition and balancing chemical equations to data normalization in relational databases [5]. We have also been very successful in developing ITSs for ill-defined, design tasks. SQL-Tutor [13] is our first ITS that teaches students how to formulate queries in SQL. The typical problem statement specifies the type of data the query is to return, and is often ambiguous. The student needs to understand the database structure and the data that is stored in the database, as well as the relational data model and the SQL constructs. The transformation for the problem statement into an SQL query is underspecified; the students are taught search strategies on examples, and need to be able to use analogies with previously solved queries to solve new ones, as well as to use the SQL syntax. EER-Tutor [5] (the early version of which was known as KERMIT [14]) teaches conceptual database design, the task previously discussed in section 1. COLLECT-UML is another ITS that teaches a design task, this time object-oriented software design using UML class diagrams [15]. In addition to teaching UML class diagrams, COLLECT-UML also teaches collaboration skills.

Our tutors are based on Constraint-Based Modeling (CBM), a domain and student modeling approach that represents domain knowledge as a set of constraints on correct solutions [16]. Constraints capture both syntax and semantics of a domain, and are very computationally efficient. In our previous work, we have argued that CBM is ideally suited to teaching ill-defined tasks [12]. Constraint-based tutors do not require the problem solver, as they can evaluate the student's solution by comparing it to a solution pre-specified by a teacher. As discussed previously, it is not unrealistic to expect the

teacher to specify one preferred solution for design tasks, as such a solution is motivated pedagogically. Therefore our tutors check the semantics of the student's solution by comparing it to the pre-specified ideal solution. At the same time, they are capable of identifying alternative solutions as correct, as constraints check for equivalent ways of solving the problem.

Goel and Pirolli [17] argue that design problems by their very nature are not amenable to rule-based solutions (as in model tracing). On the other hand, constraints are extremely suitable for representing design solutions: they are declarative, non-directional, and can describe partial or incomplete solutions. A constraint set specifies all conditions that have to be simultaneously satisfied without restricting how they are satisfied. Therefore, the ITS performs the same process the student needs to perform in order to evaluate his/her solution – apply domain constraints to it.

4. How can ITSs support learning in IDIT?

We argue that CBM is suitable for use in both well- and ill-defined domains/tasks. We have discussed examples of constraint-based tutoring systems that work in the two quadrants in Figure 1 corresponding to well-defined domains (WDWT and WDIT), with well- or ill-defined tasks. Can CBM also be used in the IDIT quadrant?

To answer this question, let us discuss an example instructional task that belongs to this quadrant. Walker et al. [18] describe one such situation: teaching intercultural competence, a task in which the student needs to explain observed cultural behaviour. This is an ill-structured task, as the start/end goals are ill-defined, there is no algorithm to use to solve the problem, and additionally there is no stopping criterion. The domain theory is also ill-defined; certain norms and rules are known about a culture involved, but there are many exceptions to them and also personal differences make the whole process very hard. Model tracing cannot be used in this case, as it is hard (or maybe even impossible) to come up with the cognitive model of the task. In [18], whatever is known about the domain is captured in terms of five dimensions: the student's discussion needs to be on topic, must be based on provided facts, needs to contain multiple perspectives and a good argument for the claims and observations. The fifth dimension allows the student to also provide novel facts to support their argument, which have not been provided in the case. Walker et al. assess student's discussion by comparing it to the model of a good discussion – in essence this model is an "ideal" solution. Since the domain and the task are both ill-defined, it is impossible to specify the ideal solution completely; in such situations, the ideal solution consist of *mandatory* elements, and other elements are optional, and depend on the personal preference.

Walker et al. [18] effectively do what constraint-based tutors do: they created a mechanism for comparing the student's solution to the ideal solution provided by the teacher. They also provide feedback to the student, saying what is good in his/her discussion, and suggesting how to improve. CBM can be applied in this case: the constraint set would consist of the syntax restrictions that must be satisfied, and the semantic constraints would check that the solution is consistent with the given problem. This implies another possible design for constraint-based tutors: the one in which the ideal solution would be replaced with a formal specification of problem requirements.

Another illustration is the domain of legal reasoning, as done in LARGO [19]. In this system, the student needs to develop a diagram reflecting the legal case. This diagram is then compared to the expert's solution, and feedback is provided about

wrong or missing elements. Architectural design also belongs to this group. If the task is to design a house with three bedrooms for a given piece of land which needs to be eco-friendly and energy efficient, there can be a set of designs which satisfy the minimal requirements. Constraints that need to be satisfied involve the problem specification and the norms for energy consumption and ecological consequences – but the designs will differ in terms of aesthetics and personal preferences of the designer. Again, the constraint set will capture the minimal requirements, and still allow for a variety of solutions. Therefore, in ill-defined domains the student has the freedom to include solution components to make the solution aesthetically pleasing or more to their preferences, and the ITS will still accept it as a good solution for the problem. It is also possible to have weights attached to constraints, with highest weights being assigned to mandatory constraints, and lower weights assigned to constraints that need not necessarily be satisfied as they correspond to optional elements.

In IDIT, more attention needs to be devoted to the feedback provided to the student. In well-defined domains, feedback generation is straightforward: the student violates some constraints, and feedback on violated domain principles is provided. In model-tracing tutors, buggy production rules provide feedback on errors, and hints can be generated on the next step the student is to take. However, in ill-defined domains, the declarative knowledge is incomplete: the constraint set consists of a set of mandatory principles and some heuristics. Therefore, the feedback mechanism needs to be sophisticated, so that feedback does not confuse the student. Walker et al. [18] provide suggestions to the student, based on prioritized dimensions they identified, which is in essence similar to the mechanism we use to select a tutorial dialogue in KERMIT [20]. If the solution is a partial one, feedback becomes even more crucial, as the ITS should discuss only the issues the student has worked on so far.

Ill-defined domains and tasks are very complex, and therefore, ITSs need to scaffold learning, by providing as much information as possible without making it trivial. The common ITS techniques can also be used in ill-defined domains (e.g. visualizing goal structure and reducing the working memory load [13], providing declarative knowledge in the form of dictionaries or on-demand help [6, 7], etc). Furthermore, the ITS can simplify the process by performing one part of the task for the student automatically [21] or by restricting the actions students can take [4]. Furthermore, solution evaluation can be replaced with presenting consequences of student actions [22] or supporting a related, but simpler task, e.g. peer review [23].

5. Conclusions

This paper proposes two important dimensions for discussing ill-definedness, which can be used to order the domains/tasks along a continuum, from well- to ill-defined. As designers of ITSs, we cannot do much about the domain definedness – it is either well- or ill-defined (in other words, the domain theory either exists or does not exist). However, the features of instructional tasks are very important. We discussed the definedness of the initial state (i.e. the problem statement) and final, goal state (i.e. the form of the solution), as well as the problem-solving procedure and the number of alternative solutions. Instructional tasks vary on all of those four factors.

In previous work, we have shown that CBM can be used effectively to support learning in well-defined domains, with either well- or ill-defined tasks. CBM can deal with ill-defined tasks, as each constraint tests a particular aspect of the solution, and

therefore supports modularity. Incremental development is supported by being able to request feedback at any time. Therefore, CBM can evaluate partial solutions: if a particular part of the solution is incomplete, the student will be informed about that.

We also show that CBM can support ill-defined domains, by discussing current research on such domains, and identifying similarities with constraint-based tutors. Constraints can capture whatever is known about the ill-defined domain and the problem specification, thus begin able to evaluate the mandatory parts of the solution. Such a tutor can provide feedback to student, while still allowing for multiple solutions differing in non-essential elements, such as aesthetical and personal preferences.

References

[1] V. Aleven, K. Ashley, C. Lynch, N. Pinkwart (eds) ITS 2006 Workshop on Intelligent Tutoring Systems for Ill-Defined Domains, 2006.

[2] V. Aleven, K. Ashley, C. Lynch, N. Pinkwart (eds) AIED 2007 Workshop on AIED applications in ill-defined domains, 2007.

[3] V. Aleven, K. Ashley, C. Lynch, N. Pinkwart (eds) ITS 2008 Workshop on Intelligent Tutoring Systems for Ill-Defined Domains: assessment and feedback in ill-defined domains, 2008.

[4] K. Koedinger, J. Anderson, W. Hadley, and M. Mark, Intelligent tutoring goes to school in the big city, *Int. Journal of Artificial Intelligence in Education* **8** (1997), 30-43.

[5] A. Mitrovic, B. Martin, P. Suraweera, Intelligent tutors for all: Constraint-based modeling methodology, systems and authoring. *IEEE Intelligent Systems*, **22** (2007), 38-45.

[6] V. Aleven, K. Koedinger, K. Cross, Tutoring answer explanation fosters learning with understanding. In S. Lajoie, M. Vivet (eds) Artificial Intelligence in Education, pp. 199-206, 1999.

[7] G. Weber, P. Brusilovsky, ELM-ART: an adaptive versatile system for Web-based instruction. *Int. J. Artificial Intelligence in Education* **12** (2001), 351-384.

[8] R. Elmasri, S. Navathe, *Fundamentals of Database Systems*. Addison Wesley, 2004.

[9] W.R. Reitman, Heuristic decision procedures, open constraints, and the structure of ill-defined problems. In: M.W. Shelly, G.L. Bryan (eds) *Human judgements and optimality*, pp. 282-315, 1964.

[10] R.N. Taylor, Nature of problem ill-structuredness: implications for problem formulation and solution. *Decision Sciences* **5** (1974), 632-643.

[11] V. Goel, P. Pirolli, The structure of design problem spaces. *Cognitive Science* **16** (1993), 395-429.

[12] S. Ohlsson, A. Mitrovic, Fidelity and Efficiency of Knowledge representations for intelligent tutoring systems. *Technology, Instruction, Cognition and Learning* **5** (2007), 101-132.

[13] A. Mitrovic, S. Ohlsson, Evaluation of a Constraint-Based Tutor for a Database Language. *Int. J. Artificial Intelligence in Education* **10** (1999) 238-256.

[14] P. Suraweera, A. Mitrovic, An Intelligent Tutoring System for Entity Relationship Modelling. *Int. J. Artificial Intelligence in Education* **14** (2004), 375-417.

[15] N. Baghaei, A. Mitrovic, W. Irvin, Problem-Solving Support in a Constraint-based Intelligent Tutoring System for UML. *Technology, Instruction, Cognition and Learning*, **4** (2006), 113-137.

[16] S. Ohlsson, Constraint-based Student Modelling. *Proc. of Student Modelling: the Key to Individualized Knowledge-based Instruction*, Springer-Verlag, Berlin, (1994) pp. 167-189.

[17] V. Goel, P. Pirolli, Motivating the notion of generic design with information processing theory: the design problem space. *AI Magazine* **10** (1988), 19-36.

[18] E. Walker, A. Ogan, V. Aleven, C. Jones, Two approaches for providing adaptive support for discussion in an ill-defined domain. In [4], pp. 1-12.

[19] N. Pinkwart, V. Aleven, K. Ashley, C. Lynch, Evaluating Legal Argument Instruction with Graphical Representations Using LARGO. Proceedings of AIED 2007, pp. 101–108.

[20] A. Weerasinghe, A. Mitrovic, Facilitating Deep Learning through Self-Explanation in an Open-ended Domain. *Knowledge-based and Intelligent Engineering Systems* **10** (2006), 3-19.

[21] M. Easterday, V. Aleven, R. Scheines, The logic of Babel: causal reasoning from conflicting sources. In [3], pp. 31-40, 2007

[22] R. Hodhod, D. Kudenko, Interactive narative and intelligent tutoring for ethics domain. In [4], pp. 13-, 21, 2008.

[23] I. Goldin, K. Ashley, R. Pinkus, Teaching case analysis through framing: prospects for an ITS in an ill-defined domain. In [2], pp. 83-91, 2008.