Research and Practice in Technology Enhanced Learning
Vol. 5, No. 2 (2010) 97–122

DOI: 10.1142/S1793206810000864

# A CONSTRAINT-BASED FURNITURE DESIGN CRITIC

YEONJOO OH* and MARK D GROSS†

*School of Architecture, Carnegie Mellon University*
*5000 Forbes Ave, CFA #211*
*Pittsburgh, Pennsylvania, 15213, USA*
**yeonjoo@cmu.edu*
†*mdgross@cmu.edu*

SUGURU ISHIZAKI

*Department of English, Carnegie Mellon University*
*5000 Forbes Ave, BPH #145D*
*Pittsburgh, Pennsylvania, 15213, USA*
*suguru@cmu.edu*

ELLEN YI-LUEN DO

*Colleges of Architecture and Computing*
*Georgia Institute of Technology, 828 W Peachtree St.*
*Atlanta, Georgia, 30332, USA*
*ellendo@gatech.edu*

This paper reports on the Furniture Design Critic. We propose a computational model of design critiquing using the program, which as a research tool helps us explain how to select critiquing methods in the consideration of critiquing conditions. Surveying the literature of architectural education, we have identified two dimensions from critiquing comments: (1) delivery types (interpretation, introduction/reminder, example, demonstration, and evaluation) and (2) communication modalities (written comments, graphical annotations, and images). This paper also presents how the Furniture Design Critic system selects particular methods by considering specific conditions such as the user's knowledge level and the interaction history between the user and the system.

*Keywords*: Design critiquing; constraint-based tutor; delivery types; communication modalities.

## 1. Introduction

Studio occupies a pedagogically important position in design education. It is the main academic course in any architecture or industrial design program. Students in a studio are subjected to a series of critiquing sessions, in which instructors offer feedback on their work. Many design researchers report that this critiquing process is an essential component in design teaching and learning (Boyer & Mitgang, 1996; Goldschmidt, 2002; Schön, 1985), but no systematic study has been conducted to

understand critiquing practice in design. Schön (1985) attempts to describe the knowledge and skills of design teaching in his book, *The Design Studio*, but much about design critiquing remains tacit and only very loosely articulated. Design studio teachers depend on experience from their own education or on intuition, which Weaver, O'Reilly & Caddick (2000) refer to as "*hit-and-miss*" teaching.

We envision a computer program that could offer effective critique to help individual students learn designing. However, our understanding of design critiquing as yet is too spotty or incomplete to realize this vision. Therefore, as a step in this direction we aim to develop a systematic framework to account for the decisions that a critic makes.

We built the Furniture Design Critic system to develop a computational model of design critiquing. The program provides a framework in which we can describe and explain how a critic might work, specifically, how a critic might select particular critiquing methods based on a variety of conditions.

The domain is secondary to our agenda: our main goal is not to build a learning system for furniture design. Rather, flat-pack furniture design provides a test case for system development to investigate design critiquing. It is an interesting domain where students encounter many structural and spatial design issues as they make stable furniture out of flat materials. For this reason, furniture making is often used as an early exercise for first-year design and architecture students. Compared to many other design domains, such as architectural design, the problem space of flat-pack furniture design is relatively small, so we need not account for the enormous body of domain knowledge that underlies the larger domain of architectural design. Still, like architectural design, flat-pack furniture design is an ill-defined and open-ended domain. Like architectural design, furniture design also entails drawing and model-making.

## 2. Research Scope

The purpose of the Furniture Design Critic program is to develop a model of critiquing that can lead to better teaching and learning of design. However, we are not — in the first place — concerned with answering the question: "What is pedagogically the best method of critiquing?" Nor are we concerned with constructing a model to predict which critiquing methods will be the most effective for which students under which conditions. That could be the topic of future research using the computational model presented here, but at this point, the software is simply a research tool — a framework in which to formulate alternative strategies of design critiquing, and implement them in a computational design environment.

Thus, our goal is different from that of most intelligent tutoring system research, and the reader who seeks here an empirical evaluation of our work will not find it. The evaluation of this research is not whether it helps design students learn more effectively or if we have programmed a good mechanism for selecting critiquing methods.

Rather, we seek to develop a computational model of design critiquing in which to articulate and describe design critiquing. The test of this research is the model's ability to represent and implement alternative critiquing strategies by selecting of critiquing methods. Certainly, this computational framework can be evaluated: we plan a three-phase evaluation to test our computational model of design critiquing. In Phase 1, we plan to test whether the program can represent a wide range of proposed critiquing strategies. In Phase 2, we will test if the program can represent real critiquing sessions. In Phase 3, finally we can test if our system can support students' learning. Here, however, we simply present the computational model we have developed, and argue through an example that it can be used to represent various strategies for design critiquing.

## 3. Related Work

### 3.1. *Critiquing systems*

A critiquing system is a tool that analyzes a design at hand and provides feedback to help a designer improve the solution (Fischer, Lemke, Mastaglio & Morch, 1991; Oh, Gross & Do, 2008). Although the researchers of critiquing systems take the concept of 'critiquing' from the field of design, they lack a clear understanding of design critiquing. Most critiquing systems have focused on detecting errors, or opportunities for offering corrective feedback. For example, Argo (Robbins & Redmiles, 1998), SEDAR (Fu, Hayes & East, 1997), TraumaTIQ (Gertner & Webber, 1998) all provide users with only negative evaluations. However, even a brief look at how one-on-one critiquing sessions function in design studios shows that current critiquing systems are quite primitive compared to what studio teachers do. (Examples will be provided in Section 4.)

Although critiquing is the predominant component in design education only a few critiquing systems for design have been built. Architecture critiquing systems such as CORENET (CORENET, 2009), ICADS (Chun & Ming-Kit Lai, 1997), and Solibri checker (Solibri.Inc., 2010) support only checking building codes (e.g. fire safety requirements). These systems focus on error checking and are not tightly integrated with design process; nor do they offer constructive feedback to improve the design at hand. Although task models have been developed in several design critiquing systems such as SEDAR (Fu *et al.*, 1997) and Argo (Robbins & Redmiles, 1998) systems, it is hard for these systems to recognize what a user is doing using their highly structured task models. Design, an ill-defined domain cannot be represented by well-structured models (Simon, 1969). Further, their task models are not a good way for the programs to identify critiquing opportunities from the designs.

In short, today's critiquing systems are not yet able to support design and design learning and they do not provide a framework in which to articulate design critiquing.

### 3.2. *Intelligent Tutoring Systems and Constraint-based Tutors (CBT)*

An intelligent tutoring system (ITS) is a program that tracks a student's actions and offers feedback. Critiquing systems and intelligent tutors are similar in that they all analyze the users' work-in-progress and provide feedback (Robbins, 1998). However, intelligent tutors differ from critiquing systems in that intelligent tutors intend to support students' learning, whereas critiquing systems help users improve their work at hand.

We are interested in intelligent tutors, because they use *Student Models* to activate Pedagogical Module that customizes the feedback for individual students. We think that we can adopt the design of intelligent tutors to articulate design critiquing, specifically, how to represent critiquing conditions (e.g. a certain designer's knowledge level) and how to select a certain set of critiquing methods in the consideration of critiquing conditions.

Intelligent tutoring systems have been developed for a variety of domains including mathematics (Anderson, Corbett, Koedinger & Pelletier, 1995), physics (VanLehn *et al.*, 2005), and database design (Mitrovic, Martin & Suraweera, 2007; Zakharov, Ohlsson & Mitrovic, 2005). However, there have been only few tutoring systems to support ill-defined domains such as design.

Recently there has been a great deal of attention on the development of intelligent tutoring systems for ill-defined domains (Aleven, Ashley, Lynch & Pinkwart, 2007), for example, legal argumentation (Pinkwart, Aleven, Ashley & Lynch, 2007). Design is an example, perhaps the canonical example, of an ill-defined domain (Simon, 1969). An often-cited characteristic of design is that it lacks well-structured domain models. A design problem seldom has a single or best solution; rather, a set of solutions is all satisfactory. Although design problems by their nature are not amenable to well-structured solutions as in the model-tracing approach (Koedinger & Anderson, 1997) most widely used for ITS development, the constraint-based approach (Mitrovic *et al.*, 2007; Ohlsson, 1994) is an appropriate choice for representing design solutions (Mitrovic & Weerasinghe, 2009). The constraint-based approach does not require a complete domain model. It also models domain knowledge using a set of constraints that specify what characteristics a solution should have. On one hand, these constraints can provide only a partial description of a solution. On the other hand, the effect of a missing constraint is highly restricted, resulting only in the tutor program failing to detect a particular error; a proposed solution can still be analyzed with other constraints. Thus, using the constraint-based approach we can incrementally develop a domain model in the Furniture Design Critic program.

These constraint-based tutors derive from Ohlsson's theory of learning from performance errors (Ohlsson, 1996). Ohlsson argues that learning occurs when students catch mistakes by themselves or when others catch mistakes for them. The fundamental assumption is that certain problem states reveal diagnostic information. This

assumption stems from the fact that one cannot develop acceptable solutions that violate domain principles. Antonija Mitrovic and her Intelligent Computer Tutoring Group (ICTG) have explored various topics in constraint-based tutoring, for example supporting a variety of tasks, enhancing student models and new strategies to deliver feedback, and developing authoring systems (ICTG, 2009).

Each constraint represents a piece of domain knowledge; it consists of a relevance condition and a satisfaction condition. The relevance condition indicates when the constraint should apply, and the satisfaction condition represents whether a certain piece of knowledge has been correctly applied. For every constraint that is deemed relevant to the student's problem, a solution must satisfy the satisfaction condition. A violated constraint indicates an opportunity to improve the proposed design so the constraint-based tutor then offers feedback regarding the violated constraint.

A constraint-based tutor records information about a student to make inferences about that student's knowledge of the domain. This *Student Model* consists of the history of all constraints that the tutor has applied to the student's design including both satisfied and violated constraints. The violated constraints indicate domain knowledge the student has evidently not yet mastered. Based on this diagnosis, the constraint-based tutor then provides feedback to support the student's learning.

### 3.3. *Critiquing methods in Intelligent Tutoring Systems and critiquing systems*

A drawback of conventional intelligent tutoring systems and critiquing systems is that they do not provide feedback using the rich range of methods that design instructors employ in studio teaching. (These methods are outlined in the following section.) Most computer-based systems focus only on pointing out errors and problems; although a number of systems support alternative delivery types in addition to negative evaluation such as argumentation (Fischer, McCall & Morch, 1989), examples (Nakakoji, Yamamoto, Suzuki, Takada & Gross, 1998), question-asking (Milik, Marshall & Mitrovic, 2006), or self-explanation (Mitrovic, 2002). One interesting system, Kermit (Suraweera & Mitrovic, 2002), offers six different levels of feedback; correct, error flag, hint, detailed hint, all errors, and solutions. The first type of feedback (correct) simply indicates whether a submitted solution is correct. Whenever the student submits a solution with errors, system advances to the next level in the sequence of error flag — hint — detailed hint. The student can request the other two levels: show me all errors and show me solutions. Although this system supports multiple methods, it does not make inferences about various characteristics of a student.

Several systems also offer feedback using multiple modalities. For example, Reading Tutor (Mostow *et al.*, 2003) combines speech and graphics (highlighting); AutoTutor (Graesser, Chipman, Haynes & Olney, 2005) combines speech with 3D simulation and facial expression; Design Evaluator (Oh, Do & Gross, 2004) combines text with graphical annotation of a 3D model; and KID (Nakakoji *et al.*, 1998)

employs text and images. Taken together these systems employ diverse methods to interact with users, but we are unaware of any single system that makes decisions, based on a student model, about when to use which method to critique. That is the focus of the system we present here.

## 4. Delivery Types and Communication Modalities

Studio instructors in architectural design use a variety of critiquing methods to convey their knowledge and professional skills (Schön, 1983). Using these methods, they deliver images, ideas, examples and actions from their own '*repertoires*' (Schön, 1983). They build up these repertoires from their experiences. Schön uses this repertoire concept to explain design critiquing: when instructors look at a student's solution, they scan their repertoires for similar situations, for example, buildings they have known, or problems they have previously encountered. The instructors not only point out errors; they also describe examples or demonstrate how to solve the problems. Feedback presented using multiple methods helps design students understand their problems better, eliminate errors from their proposed solutions, and construct their own repertoires (Schön, 1983; Uluoglu, 2000).

We have identified two dimensions of critiquing comments: delivery types and communication modalities, because this helps us to describe a variety of critiquing activities. For example, a studio teacher verbally introduces an idea, whereas another teacher demonstrates a plausible design solution using verbal expressions and graphic annotations. Examples of these delivery types and communication modalities used in design studios will be presented in the following sections.

### 4.1. *Delivery types*

Uluoglu (2000) and Bailey (2004) both analyze critiquing sessions in architecture studios and identify diverse 'delivery types'. These include (1) ***interpreting*** students' design solutions, (2) ***introducing*** new ideas or approaches/***reminders*** of them, (3) description of existing ***examples*** or precedents, (4) ***demonstrating*** potential solutions or other design actions, and (5) ***evaluating*** (positive or negative) of the students' solutions. We examined critiquing sessions in architecture design studio (Wampler, 2002) to see how these delivery types are used. Table 1 shows the examples of these different delivery types used in the critiquing session of Wampler's design studio.

Although a studio teacher intends to deliver the same content, it may be transmitted using different delivery types (Uluoglu, 2000). For example, the teacher leads his/her student to consider the path of the sun. In this case, the teacher's intention is same, but s/he can deliver feedback differently according to which delivery type s/he selects. If a critique presented using the demonstration delivery type shows how to consider the path of the sun by rearranging rooms or moving the positions of windows. In contrast, a critique presented using introduction/reminder delivery type provides a piece of knowledge, which is that it is a good idea for the student

to think about the path of the sun. However, the teacher does not show how to use this knowledge.

Table 1. Feedback instances of five delivery types.

| Delivery Types | Feedback Instances |
|---|---|
| Interpretation | "Your building only gets light into this level (pointing to the bottom window on the physical model)." |
| Introduction/Reminder | "Have you thought about the path of the sun over a day and over the year?" |
| Example | "Le Corbusier's building has a similar concept. Look at the windows of his chapel at Ronchamp." |
| Demonstration | "You need to make a form here. You need to do something here (drawing a line that represents a wall)." |
| Evaluation | "You take the rough form into something more precise. . . . which is good." "No good, horrible — it just ruins the whole idea." |

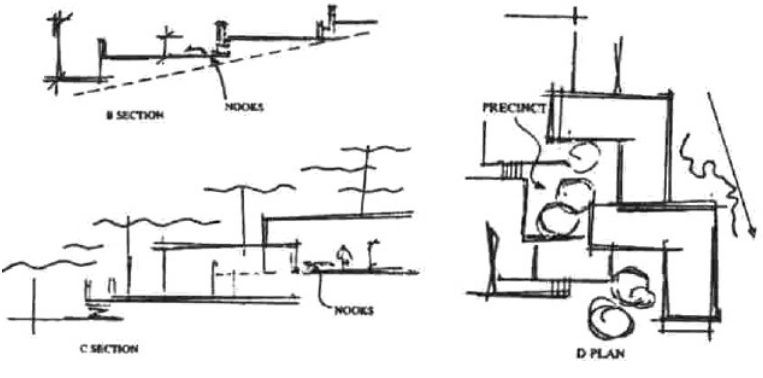
The choice of delivery types is important because it may influence a student's subsequent actions and hence learning. For example, when a teacher offers an example, a student may realize that reference to the given precedent is helpful to deal with the new design situations and attempt to adapt it to fit the work at hand. When the teacher points out errors, the student may fix them. The use of different delivery types in critiquing may lead to different reasoning and thereby further promote different learning.

### 4.2. *Communication modalities*

Design studio learning embraces numerous forms of representation such as written and graphical. Studio teachers also use these forms to communicate with their students. These forms correspond to communication modalities. These forms of communication are important, because drawing, not only verbal expressions, is an important tool to develop and communicate design solutions.

Communication modalities mean the ways that the design knowledge is presented, such as through speech, text, graphical annotation, and images. The primary modality in all face-to-face critiquing sessions is speech — teachers always talk. Studio teachers also make brief notes as they draw, or annotate their students' sketches. Although these notes are terse, they help students remember the spoken feedback. Design teachers often use drawings to describe their design ideas and demonstrate alternative solutions, ranging from abstract diagrams to representational forms. Schön (1985) and Anthony (1991) both note that critiques presented in multiple modalities work together and help students understand the intentions of their instructors. Figure 1(a) shows an everyday scene where a studio instructor (Professor Jan Wampler at MIT) sketches on a student's drawing, while offering feedback verbally (Wampler, 2002). He places tracing paper on top of the student's drawing and draws over it to suggest an alternative design. Figure 1(b) presents sketches and a brief notes that a studio instructor has made (Schön, 1985).

(a)




(b)

Figure 1. Communication Modalities: (a) speech + drawing (source: MIT Open Courseware (Wampler, 2002)); (b) drawing + text (source: The Design Studio (Schön, 1985)).

### 4.3. *Why selecting a certain critiquing method is important*

Individual critiquing situations are all different. For example, individual students are different; their knowledge levels, strengths, and weaknesses are all varying. Studio teachers cannot deal with these different students in the same way. Let us provide a simple example. A studio teacher and a design student with no previous design experience are working together in a one-on-one critiquing session in an architecture design studio. The studio teacher provides an existing building as an example to help the student take the ideas from the building and apply them into the student's current design: "*Do you know* (*architect*) *Steven Holl's chapel at Seattle University? How the building is placed in the site? Holl controls light with various shaped windows and the irregular shapes of the roof. Le Corbusier used a similar solution at Ronchamps, and Holl adopted Le Corbusier's design. They both designed the windows to control the quality of light; color, direction, and shape.*"

However, the student looks puzzled. The student cannot use the given example in his/her design, because s/he does not know how to apply the example to the current design and how to represent and explore the design ideas by making drawings. Thus, providing the example without graphical annotations is not a good choice of critiquing methods for this particular student. Instead, the teacher can demonstrate how to design his/her windows differently by dividing the roof into several different masses and by inclining a window to allow sunlight to enter the living room from a particular direction. As the teacher verbally demonstrates, the teacher draws simple perspective diagrams and section drawings. The student now understands how to use the ideas that the teacher has offered and improves his/her design using drawings. In this case, the demonstration with graphical annotation is the right choice of critiquing method for this designer, because the novice designers often cannot employ the learned knowledge to develop their own designs (Uluoglu, 2000). Thus, the choice of appropriate critiquing methods is important to help students improve their work at hand and further learn designing.

## 5. Furniture Design Critic

Motivated by the richness of critiquing in architectural design studio and the lack of understanding of design critiquing, specifically context-sensitive critiquing, we built a constraint-based design critic program. This Furniture Design Critic program offers students feedback using five delivery types (interpretation, introduction/reminder, example, demonstration, and evaluation) and three communication modalities (written comments, graphical annotations, and images). Based on the diagnosis of the student's solution, the model of the student, prior performance, and the criticism that the student has previously received the Furniture Design Critic selects a delivery type and modality with which to present a critique.

### 5.1. *System architecture*

The Furniture Design Critic is written in MCL (Macintosh Common Lisp) using OpenGL to provide 3D models and the Lisa (Lisp-based Intelligent Software Agent) production rule system to reason about a proposed furniture design using previously stored constraints. The Furniture Design Critic comprises several components: a *Construction Interface*, *Parser*, *Pattern Matcher*, *Design Constraints*, *Critiquing Rules*, *User Model*, *Pedagogical Module*, and *Critic Presenter*. Figure 2 shows these components, their relationships, and the information flow among them. This section follows the process shown in Figure 2 to describe what individual components do and how the system works.

### 5.2. *Construction interface and parser*

A designer starts to design by sketching an axonometric diagram in the *Construction Interface* using a stylus and a digitizing tablet. The program records all designer's

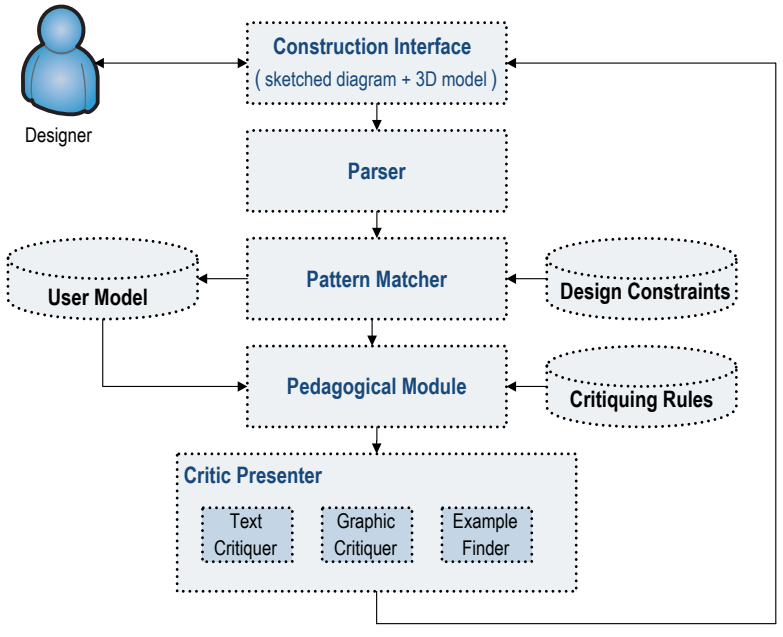



Figure 2. Furniture Design Critic system architecture in the iterative construction-critiquing-repair cycle.

sketched marks, identifies the Cartesian coordinate system that is implicit in the drawing, and then generates a 3D model.

The *Parser* examines the sketched diagram and the 3D model and produces two kinds of data: a list of individual parts, their properties (e.g. $x$-length, plane, 3D coordinate data, joints, etc.) and the configuration of the parts (e.g. parallel, between, top-of, jointing, distance, etc.). Based on these data, the program recognizes the function of a furniture piece (e.g. table and chair) and the function of each part (e.g. top, shelf, and side). Specifically, the program has a list of representations for particular furniture pieces. It compares them against the symbolic representation of designer's furniture using the *Pattern Matcher*. If matched, the program recognizes what the designed furniture is and what functions individual parts have. The *Parser* stores this symbolic representation of the designed furniture in a text file. This symbolic representation will be used to identify which constraint is satisfied or violated.

### 5.3. *Design constraints*

The program uses a set of *Design Constraints* that represent principles that furniture designers need to know. Furniture Design Critic uses two types of constraints: 43 structural constraints and 57 functional constraints. The structural constraints specify valid configuration of furniture parts and are used to identify structural problematic parts in designers' solutions. They vary from simple constraints such

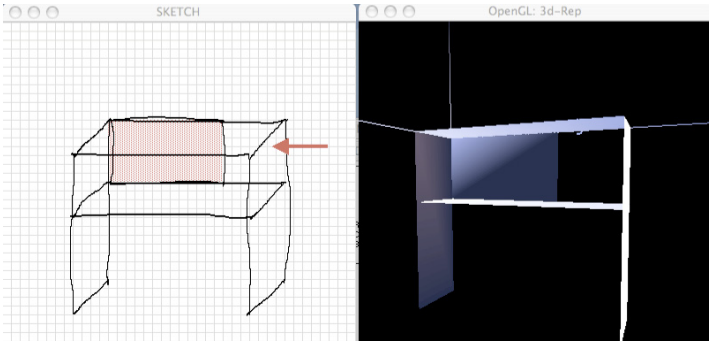
as "a long shelf must be supported from the middle", to more complex constraints such as "a table's brace must be placed carefully to allow enough leg space." Simple constraints only deal with a single piece of furniture. Complex constraints are used to identify problems in related pieces of furniture, such as a table and a chair. The leg space constraint mentioned above checks whether a table provides enough leg space by comparing the position of a table's brace, with the width and the height of the chair that a designer has previously designed and stored. Functional constraints specify the functions of certain parts or a whole piece of furniture. For example, "a chair may have armrest" and "a horizontal bracing part parallel to the chair seat may be used as a shelf."

Each constraint consists of a relevance condition, a satisfaction condition, its importance, written comments using five different delivery types, function calls to present feedback using other two communication modalities. The importance of a constraint is represented as an integer value between 1 and 3 based on its potential influence of a violation on the stability of the whole furniture piece. Level 1 constraints are important for stability. If a level 1 constraint is violated, the furniture piece will fall down. For example, "a bookcase must have a back in order to support lateral loads." Level 2 constraints are influential on the furniture design but not critical for stability. For example, "the height of the tabletop must be higher than the height of the chair seat and the desirable difference is from 45% to 65% of the height of the chair seat." The constraints with low importance (level 3) refer to minor issues, although they are still relevant to the furniture designs. For example, "having only one armrest breaks symmetry on the chair design." When these constraints are violated twice, the program removes these constraints from further consideration as these constraint violations do not critically influence stability.

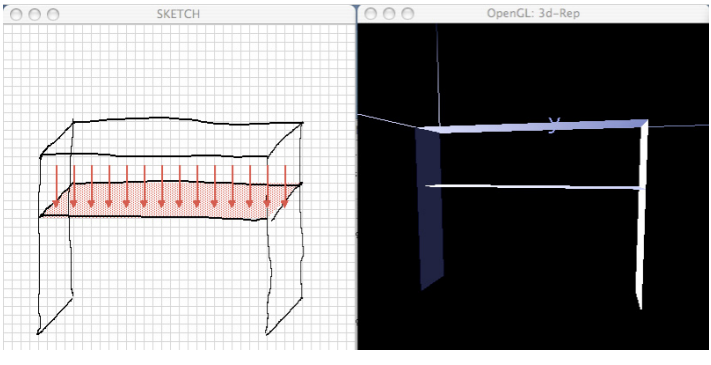
The constraint satisfaction conditions require judgment of furniture design that may not seem easily automated. However, we have implemented constraints that can determine, for example, whether a corner of a table is unsupported by comparing the positions of the legs relative to the corners of the table. If this distance is too large proportionally to the dimension of the table, the corner is considered unsupported. Although we have implemented each satisfaction condition individually, a more general way to address this challenge is to employ a physics engine that subjects the furniture to simulated real-world forces.

Each constraint data structure stores two items relevant to offering feedback in multiple methods: *critique-delivery-types*, and *critique-modalities*. The *critique-delivery-types* item stores pre-defined written comments for the constraint in five different delivery types. For example, the constraint in a bookcase design checks whether a back part is large enough to support lateral loads (see Figure 3(a)). The *critique-delivery-types* item stores five different written comments:

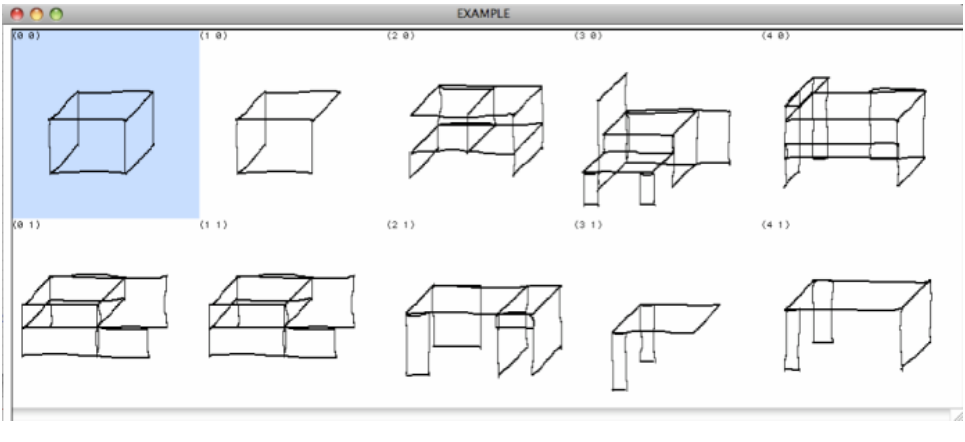
**Interpretation** – "Your bookcase is composed of two sides, a shelf, a top and a back. The two sides and the back are vertical structural components for loads."




(a)




(b)




(c)

Figure 3. (a) The program highlights the problematic part (the back) in red and draws an arrow to indicate lateral loads. (b) The program draws arrows to indicate vertical loads placed on the shelf. (c) The program retrieves relevant examples and presents them.

**Introduction/Reminder** – "Is your back part big enough to support lateral loads?"
**Example** – "Look at other bookcases with backs."
**Demonstration** – "Make the back part bigger as shown (drawing a bigger back part)."
**Evaluation** – "The back is an important part to support lateral loads, but it's too small. Your bookcase is structurally unstable for lateral loads"

The *critique-modalities* item stores a list of calls to routines that deliver feedback in different communication modalities. The Furniture Design Critic delivers feedback using the selected communication modalities by executing these routines: graphic annotations, e.g. painting parts red that violate a constraint (Figure 3(a)); displaying graphic icons such as arrows to indicate a load placed on a furniture part (Figure 3(b)); and retrieving and presenting images of relevant examples (Figure 3(c)).

### 5.4. *Pattern Matcher*

The *Pattern Matcher* compares the symbolic representation of the design against the *Design Constraints* in order to detect critiquing opportunities. For example, the bookcase design in Table 2 violates the stored constraint that "a long shelf of a bookcase must be supported from the middle". The pseudo-code and diagrams show the constraint that the design has violated.

### 5.5. *User model*

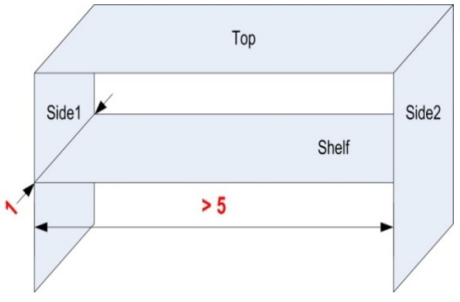
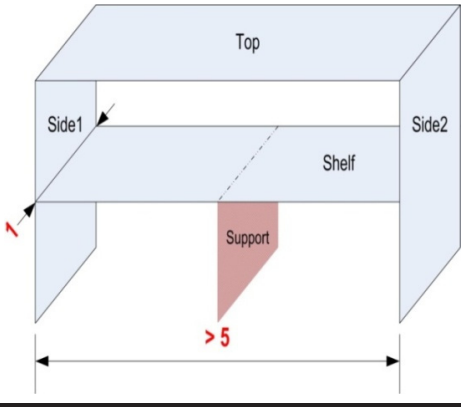
The Furniture Design Critic stores two types of *User Model*: a short-term and a long-term user model. The short-term user model stores the reasoning outputs of the *Pattern Matcher*, namely which constraints are satisfied or violated in the current critiquing session.

The long-term user model is similar to the short-term user model, but it stores the history of all violated and satisfied constraints over multiple critiquing sessions. Using this history the *Pedagogical Module* makes inferences about how much a designer knows about flat-pack furniture design; the designer's specific weaknesses, and which critiquing methods are effective for helping the designer (see Section 5.7).

### 5.6. *History of states*

The Furniture Design Critic keeps track of the history of *states* of the system. A *state* represents a critiquing condition: (1) the constraint that was violated, (2) the selected set of critiquing methods, and (3) whether the selected critiquing methods were effective. This *state* is captured when the program offers feedback. This is stored as the part of the *User Model*; although it does not describe characteristics of a designer explicitly, the system employs it when selecting critiquing methods.

Table 2. A violated constraint example.

| | |
|---|---|
| If the designed furniture is a bookcase<br>with a long shelf<br>Top and Shelf are **parallel**<br>Side1 and Side 2 are **parallel**<br>Shelf is **between** Side1 and Side2<br>Side1 and Shelf are **jointed**<br>Side2 and Shelf are **jointed**<br>Shelf is **long** (length > width∗5) |  |
| then a shelf must be supported from the middle<br>Shelf is **on top of** Support |  |

For example, when a particular set of methods have been used repeatedly in two previous *states* and the critiques have been unsuccessful both times, the program will select alternative critiquing methods to offer feedback.

### 5.7. *Pedagogical module*

The main task of the *Pedagogical Module* is to select particular sets of delivery types and modalities by considering characteristics of the critiquing situation such as the inferred data about a designer and the history of *states* (see Section 5.6). The *Pedagogical Module* makes these decisions about the critiquing methods by applying *Critiquing Rules* (see Section 5.8).

The *Pedagogical Module* takes as input (1) data of violated constraints from the short-term user model, (2) data of the *User Model* and (3) *Critiquing Rules*. In other words, it considers the violated constraints and the *User Model* and chooses a particular critiquing method by applying the *Critiquing Rules* (Figure 4). In sequence, the *Pedagogical Module* (1) selects which feedback should be offered first; (2) makes an inference about a designer using the data of *User Model*; and (3) selects a certain set of critiquing methods by applying *Critiquing Rules*.

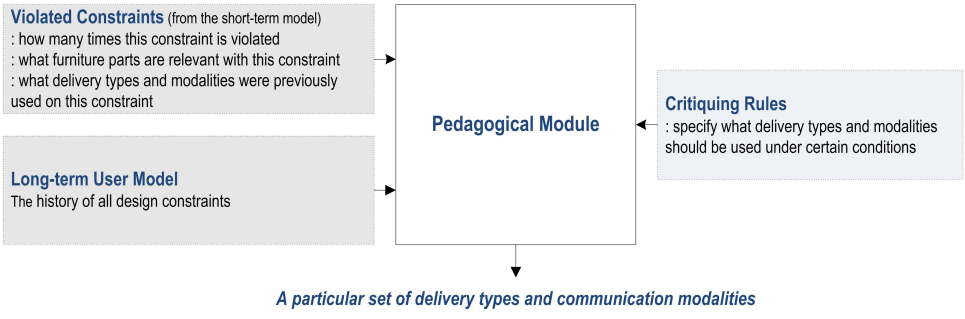



Figure 4. *Pedagogical Module* and the data involved for selecting critiquing methods.

### 5.7.1. *Selecting which feedback should be offered first*

If a design violates more than one constraint the *Pedagogical Module* must decide in what order to deliver feedback. It addresses more important issues first. If several constraints have same importance, the program prefers any constraint that has previously been violated.

### 5.7.2. *Making an inference about a designer using the data of User Model*

The *Pedagogical Module* takes the history of all constraints including the violated and satisfied (from the *User Model*) and infers (1) how much the designer knows about furniture design; (2) which type of constraints that the designer has trouble with and what types of constraints the designer handles properly; and (3) which critiquing methods work well for that designer.

Let's look at how the *Pedagogical Module* makes an inference about the designer. $\mathbf{R_{Violated}}$ in Eq. (5.1) represents how much a designer does (not) understand about the design space represented by constraints. That is, the higher $\mathbf{R_{Violated}}$, the less the designer knows. $\mathbf{R_{ViolatedCritical}}$ in Eq. (5.2) represents the designer's ignorance of important constraints. The higher $\mathbf{R_{ViolatedCritical}}$, the less a designer knows about the fundamental knowledge of the design domain. Critical constraints with high importance (level 1) should be satisfied for designers to develop stable furniture.

$$\mathbf{R_{Violated}} = (\text{number of constraints violated/number of all constraints the program knows}) \tag{5.1}$$

$$\mathbf{R_{ViolatedCritical}} = (\text{number of critical constraints violated/number of constraints violated}) \tag{5.2}$$

The program infers a designer's knowledge level based on the history of all design constraints that the program knows. The program represents the designer's knowledge level by a pair of ratios ($\mathbf{R_{Violated}}$, $\mathbf{R_{ViolatedCritical}}$). When these two ratios are zero (0, 0), then the designer has violated no constraints and therefore

has mastered all the domain knowledge that is stored as constraints in the system. (As the knowledge level is unknown at first, these ratios are initially (nil, nil). The program distinguishes this initial level from the perfect mastery level of (0, 0).)

The system computes each designer's weaknesses with respect to structure and function. Weakness of structural knowledge is represented by the ratio $\mathbf{R}_{\mathbf{WeaknessStructural}}$ in Eq. (5.3). Weakness of functional knowledge is represented by the ratio $\mathbf{R}_{\mathbf{WeaknessFunctional}}$ in Eq. (5.4). If $\mathbf{R}_{\mathbf{WeaknessStructural}}$ is high, the program recognizes that the designer is weak in applying structural constraints to his/her designs.

$$\mathbf{R}_{\mathbf{WeaknessStructural}} = (\text{number of structural constraints violated/number of structural constraints relevant to the current design task}) \tag{5.3}$$

$$\mathbf{R}_{\mathbf{WeaknessFunctional}} = (\text{number of functional constraints violated/number of functional constraints relevant to the current design task}) \tag{5.4}$$

The Furniture Design Critic infers which effective critiquing methods work well for each designer based on its long-term user model. When a designer repeatedly succeeds in response to the critiques offered using a certain method, the program stores that method as an effective critiquing method for that designer.

We decouple the judgments about the designer from the *User Model*, because of the program's scalability. The current state of this program infers the designer's weaknesses in the structural and functional knowledge, overall knowledge level, and effective critiquing methods. For example, if we add another kind of weakness in aesthetic knowledge — aesthetic constraints, the program can takes the history of all constraints and computes the designer's aesthetic weakness in the *Pedagogical Module* by defining a new *Pedagogical Rule*. We do not need to change the *User Model* to judge the designer's aesthetic weakness. Specifically, if the ratio, (e.g. number of aesthetic constraints violated/number of aesthetic constraints relevant to the current design task) is higher than a certain number while executing the *Critiquing Rule*, it indicates that the designer is weak in that aesthetic category of knowledge. This design of the program enables us as the system designer to easily scale up this program for supporting a larger design domain or other design domains.

### 5.7.3. *Selecting a set of delivery types and communication modalities*

Based on the judgments about the designer, the *Pedagogical Module* selects delivery types and communication modalities. It creates a text file that stores a representation of these critiquing conditions including (1) the violated constraint; (2) data of the *User Model* such as the history of the *States*; and (3) the data that represent

the designer's characteristics such as knowledge levels, weaknesses and effective critiquing methods. It uses this file to make a decision about which critiquing methods are appropriate in each situation.

To select an appropriate set of critiquing methods, the *Pedagogical Module* applies *Critiquing Rules* into the critiquing condition representation. It executes actions such as selecting critiquing method candidates and adding a delivery type, if the data of the text file satisfy the condition of a *Critiquing Rule*.

### 5.8. *Critiquing Rules*

Currently the program has 76 *Critiquing Rules* that specify which delivery types and communication modalities to use under what conditions. These rules are formulated based on the literature of design and design education, and from informal sessions in which design studio instructors discussed their teaching techniques. For example, for novice designers, directive feedback such as demonstration and evaluation are better than facilitative feedback such as examples, because novices often experience difficulty using abstract ideas. Here is another example. When the piece of knowledge is introduced, but a designer cannot apply it to his/her design, it is a good idea to show how to use the idea by demonstrating a plausible solution (Uluoglu, 2000). Although it would be interesting and useful, we did not conduct a formal evaluation of the critiquing rules, as this is beyond the scope of our work.

The following pseudo-code shows a *Critiquing Rule* that selects delivery type candidates, (demonstration, evaluation, (demonstration, evaluation)), when the data that the *Pedagogical Module* takes satisfies both conditions ($\mathbf{R_{Violated}} \geq 0.4$) and ($\mathbf{R_{ViolatedCritical}} \geq 0.5$). These conditions mean that the designer's knowledge level is quite low.

**Critiquing Rule# DeliveryTypes-Designer-LowKnowledgeLevel**

[**conditions**] if ($\mathbf{R_{Violated}} \geq 0.4$) and ($\mathbf{R_{ViolatedCritical}} \geq 0.5$)

[**actions**] select delivery type candidates

(***delivery-type-candidates*** (demonstration, evaluation, (demonstration, evaluation))).

### 5.9. *Critic presenter*

Once the*Pedagogical Module* selects a critiquing method, the *Critic Presenter* activates one or more helper components to present the critique. The *Critic Presenter* has three helpers: (1) a Text Critic, (2) a Graphic Critic, and (3) an Example Finder. The Text Critic finds the written comments associated with a violated constraint and present them below designer's diagram. For example, if the selected delivery types are introduction/reminder and evaluation, the Text Critic selects and displays the second and fifth written comments from the *critique-delivery-types* item of a violated constraint (see the *critique-delivery-types* item example presented in

Section 5.3). The Graphic Critic highlights relevant furniture parts, draws graphical annotations on a designer's diagram, and presents images by executing function calls stored in the *critique-modalities item*. The Example Finder selects relevant examples from a library by comparing the symbolic representation of the current design against those of the stored examples. Here, the library is a collection of the designs stored previously in the form of symbolic representations, which contains furniture parts, spatial configurations among them, and a list of constraints that are violated or satisfied. For example, if a designer is designing a table, the program retrieves only the designs considered as tables by analyzing the spatial configurations or the list of violated/satisfied constraints (Figure 3(c)).

We summarize how our program works before presenting an example scenario. A designer makes a design diagram to develop a piece of furniture. The program generates a 3D model based on the sketched diagram. Based on this diagram and 3D model, the Furniture Design Critic recognizes his/her design and finds critiquing opportunities by comparing this design against design constraints. The *Pedagogical Module* makes decisions about critiquing methods by considering the *User Model* and the history of *states*. The *Critic Presenter* then presents feedback using the selected methods.

## 6. Example

An example represents a particular selection mechanism that the *Pedagogical* uses to select delivery types and communication modalities. The reason we present this example here is to present a plausible way to describe design critiquing, specifically, how to select critiquing methods in the consideration of critiquing conditions. However, we do not claim that this selection mechanism should be followed or is correct. Rather, through this example, we intend to show that different selection mechanisms can be developed by changing the *Critiquing Rules*. It is because that the *Critiquing Rules* decides the performance of the *Pedagogical Module*. Also these *Critiquing Rules* are easily modified and added.

### 6.1. *A selection mechanism of the Pedagogical Module*

Let us now examine how the *Pedagogical Module* reasons about the selection of a particular set of critiquing methods. Figure 5 shows the four-step reasoning process by which the *Pedagogical Module* selects a particular set of delivery types and communication modalities. By applying the *Critiquing Rules* in sequence, the program selects candidate critiquing methods and then in each step boxes the candidates.

In Step 1, the *Pedagogical Module* looks at the *User Model*, which represents the designer's knowledge level, strengths and weaknesses, and the methods of critiquing that have proven effective. These characteristics lead the program to select critiquing method candidates.

If the *User Model* can provide information for selecting critiquing methods the program follows Steps 2, 3, and 4. However, if the *User Model* does not provide

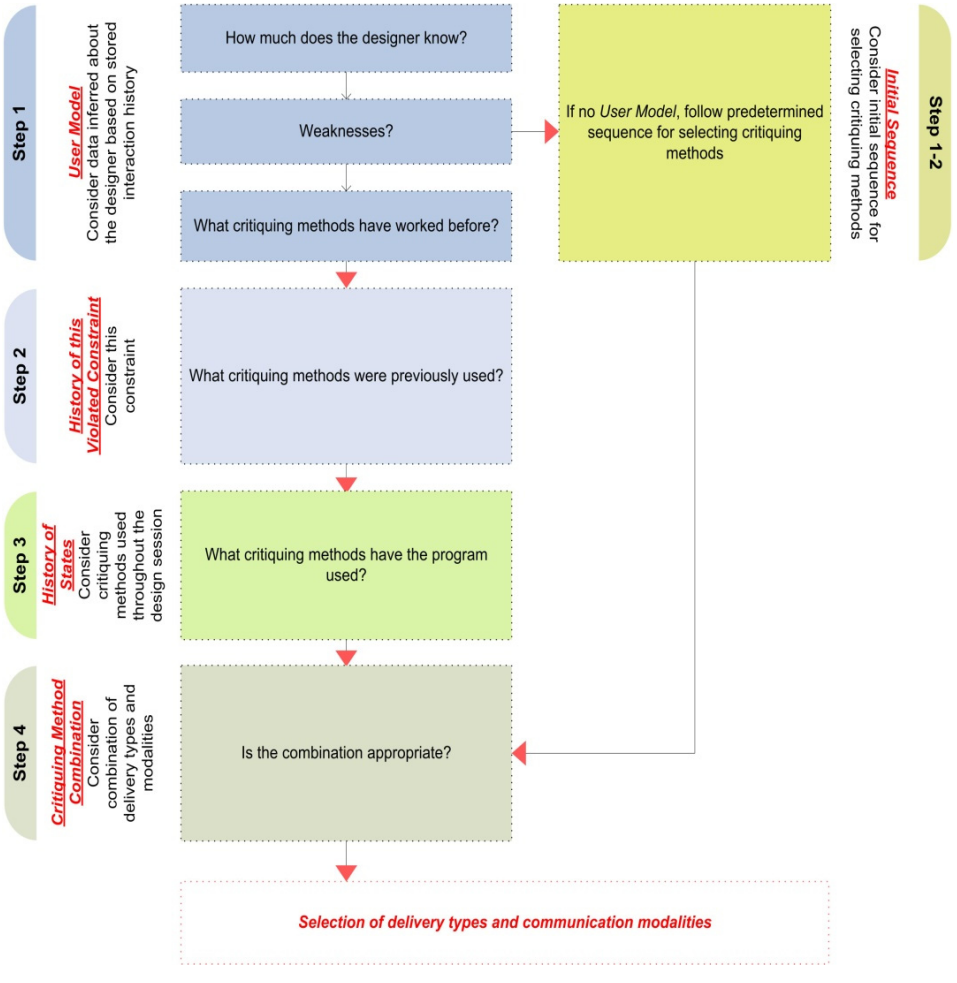



Figure 5. The reasoning process to determine a particular set of delivery types and communication modalities.

information (for example, if the user has no history with the program), then the program simply chooses a predetermined sequence of methods. In this case, for delivery type, the program follows the sequence: interpretation, introduction, example, demonstration, and evaluation. For communication modality, it follows the sequence: written comments, graphical annotation, written comments + graphical annotation, written comments + images, written comments + graphical annotation + images.

We chose these sequences to first offer critiques using facilitative delivery types (e.g. interpretation, introduction or example), because this feedback can prompt a designer to think about and improve the design. When the designer is unable to

benefit from facilitative critiques, the program changes to directive feedback (e.g. demonstration or evaluation).

In Step 2 the program considers what critiquing methods have been used previously to address the violated constraint. The program removes the methods used from the candidate set to avoid repeatedly offering the same form of feedback on the same constraint violation. If all candidates have already been used for the constraint, then the program adds a new method to the candidates from among the methods that have not been used.

In Step 3 the program considers what critiquing methods have been used in all previous *states* regardless of the current constraint violation. For example, when a set of critiquing methods has been used more than twice in the previous *states*, the program selects a different critiquing method to avoid giving feedback in the same way. For example, when the program believes that the most effective critiquing method for the designer is evaluation; it will keep offering evaluative feedback. It is reasonable to communicate with the designer using only methods that have worked. However, this might also hinder the designer's opportunities to reflect on the design in different ways. In this step, therefore, the program experimentally attempts to communicate with the designer in alternative ways by considering the previous *states*. In other words, this step performs an experiment to determine whether these alternative methods can help the designer improve the design, although the *Pedagogical Module* has not selected these methods based on the critiquing situation, such as the data of the *User Model*. The result of this experiment is stored in the history of *states* and then will influence the next selection.

Finally, in Step 4, the program considers the combination between delivery type and communication modality candidates to determine whether it is appropriate. For example, if the program has selected demonstration as a delivery type, it checks whether graphical annotation or image is also selected as a communication modality. Graphical annotation and image can facilitate the designer's understanding of the demonstrated ideas. The program then returns the selected critiquing methods to the *Critic Presenter*, which will present the critiques.

### 6.2. *Scenario*

The following scenario illustrates how the Furniture Design Critic works, especially, the reasoning process of the *Pedagogical Module*. It demonstrates how the program selects a particular set of delivery types and communication modalities in a given situation.

A designer (Claire) draws a diagram while developing her furniture (Figure 6). The program offers three written comments: "A user cannot keep upright posture while sitting on the chair. The center of gravity can move around"; "You can add legs to support the unsupported corners. They will improve stability"; "Your chair is unstable when a load is placed on the unsupported corner(s)." The program also annotates Claire's drawing, adding missing parts of the chair (legs) to her diagram.

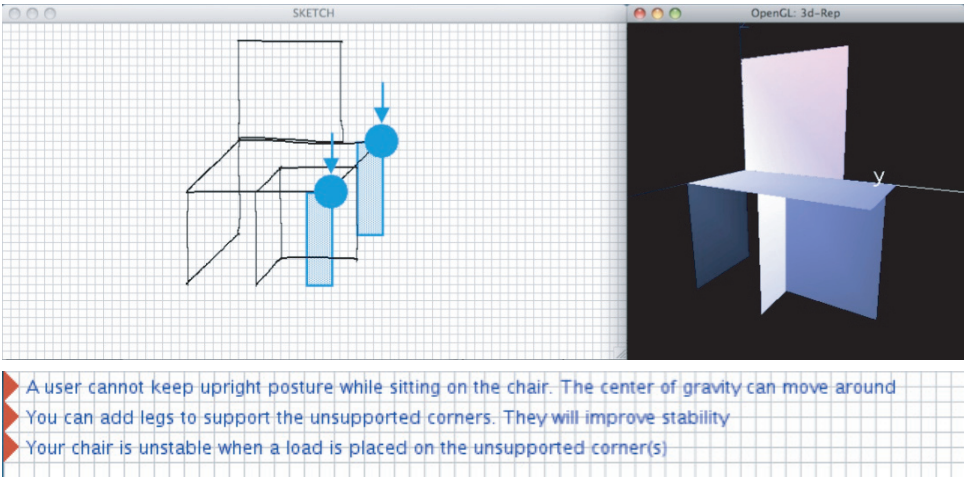



Figure 6. The program presents critiques using the selected critiquing methods: <[intro/reminder, demonstration, evaluation] and [written comments, graphical annotations]>.

We now examine in detail how Furniture Design Critic selects these delivery types and communication modalities to present the critique shown in Figure 7 (the program follows the sequence of steps shown in Figure 2).

First, the program decides what Claire is designing by analyzing the spatial relationships of the furniture parts. In her design, three vertical parts (the legs) support a horizontal part (the seat) and another vertical part (the back) is placed on the horizontal part (seat). Based on these relationships, the program recognizes that she is designing a chair, so the program selects a set of design constraints pertaining to chair design.

Next, comparing these spatial relationships against the design constraints, the program identifies opportunities to critique Claire's design. Each constraint represents a design principle defined in terms of spatial relationships. Here it detects several critiquing opportunities: the seat needs more vertical supports, the legs need a brace, and the chair could have armrests. Each of these is a relevant constraint that the design violates. In this example, the program chooses the constraint that a seat needs more vertical supports for its stability.

Now the *Pedagogical Module* considers this chosen violated constraint in the context of Claire's *User Model* and all previous *states* to select an appropriate set of critiquing methods. It follows the four-step reasoning process in Figure 5, which presents what the *Pedagogical Module* considers to determine a particular set of delivery types and communication modalities. Figure 7 shows how the *Pedagogical Module* reasons about the selection of critiquing methods with concrete data in Claire's case. Each box in Figure 7 represents a step of the reasoning process. Figure 7 summarizes what critiquing methods are selected as candidates and why

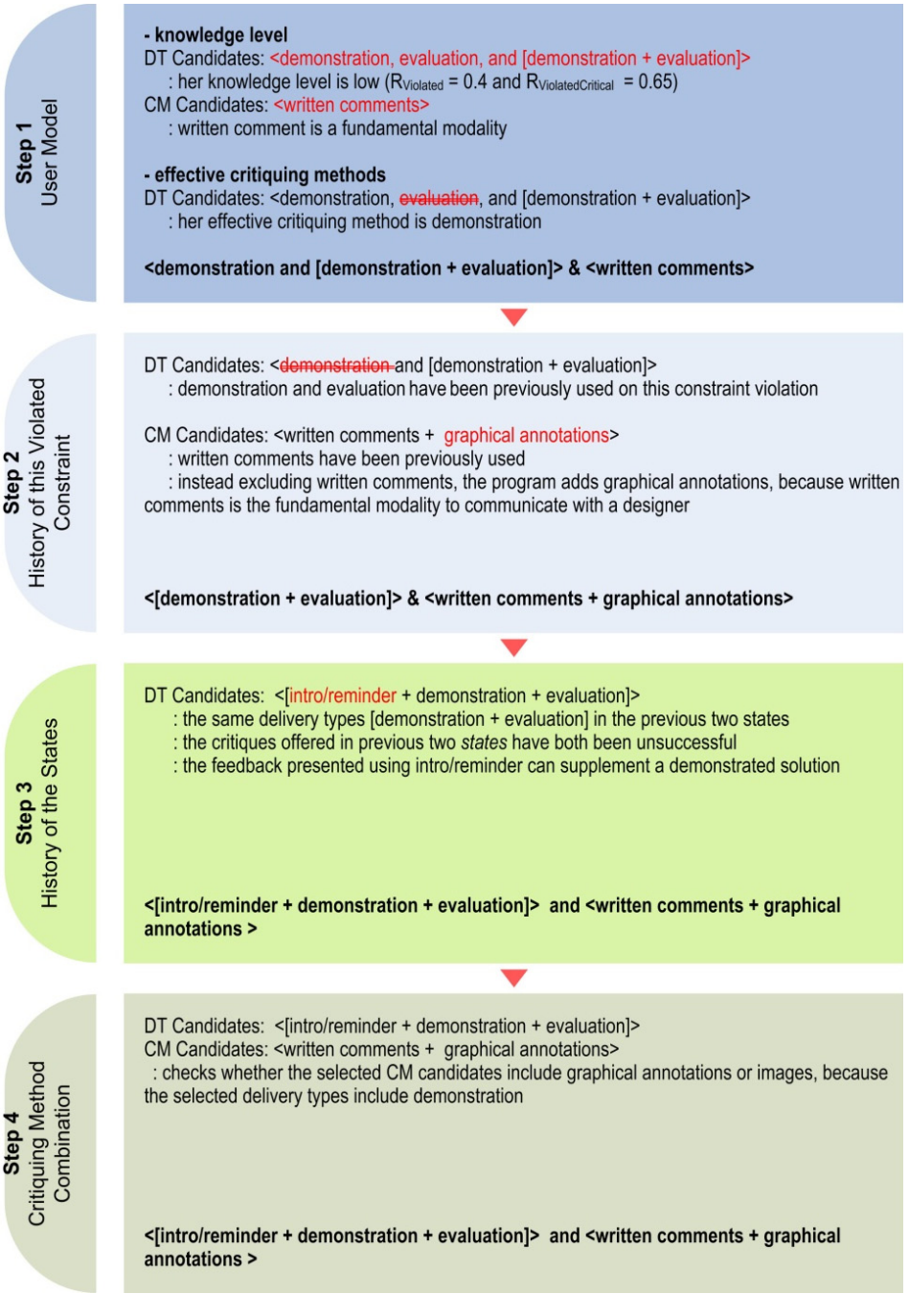



Figure 7. The reasoning process of the *Pedagogical Module* in Claire's case (DT: delivery type, CM: communication modality). DTs and CMs in strikethrough indicate candidates that the program has eliminated. DTs and CMs in red indicate candidates the program has added.

they are selected with relevant data, such as Claire's knowledge level and history of *states*.

In Step 1, it analyzes Claire's *User Model*, knowledge level, weaknesses, and effective critiquing methods. Her knowledge level is quite low ($\mathbf{R_{Violated}} = 0.4$ and $\mathbf{R_{ViolatedCritical}} = 0.65$), so the program selects three delivery type candidates: demonstration, evaluation, and [demonstration + evaluation]. The program starts with written comment as a communication modality candidate, because the program always uses written comments as a fundamental modality, which means that a delivery type – written comment – is always selected. The program looks at the critiquing method that has been effective with Claire, which is demonstration. It selects only delivery type candidates that include demonstration, so the current candidates are <demonstration, [demonstration + evaluation]> and <written comments>.

In Step 2, the program looks at what critiquing methods have been used on the previous violations of the constraint in question. Delivery types demonstration and evaluation and communication modalities written comments and images were used. The program eliminates demonstration from the candidates to avoid offering feedback in the same way. However it does not delete written comments, because written comments is the fundamental modality to communicate with a designer. Instead, it adds graphical annotations to the set of communication modality candidates. At this point, the candidates are <[demonstration + evaluation]> and <written comments + graphical annotations>.

In Step 3 the program analyzes what critiquing methods were previously used in the stored *states*. Furniture Design Critic has selected the same delivery types [demonstration + evaluation] twice. A *Critiquing Rule* says that the same methods may not be repeated more than twice, when the critiques offered in previous two *states* have both been unsuccessful. Therefore, the program adds another delivery type, intro/reminder, because the feedback presented using intro/reminder can supplement a demonstrated solution (demonstration) and the problematic part identified with evaluation. This combination intends to help Claire to understand the idea that lies behind the demonstrated course of action and evaluation. At this step, the candidates are <[intro/reminder + demonstration + evaluation]> and <written comments + graphical annotations>.

Finally in Step 4 the program considers whether the combination among delivery type and communication modality candidates is appropriate. It checks whether the selected communication modality candidate includes graphical Annotations or images, because the selected delivery types include demonstration following a *Critiquing Rule*. Graphical annotations or images facilitate Claire's understanding of the demonstrated information, because it provides Claire with a concrete idea about how to resolve the raised issue. The program thinks that the selected method candidates make appropriate combination among delivery type and communication modality candidate, because the candidates include graphical annotations already.

The final selected critiquing methods are [intro/reminder + demonstration + evaluation] and [written comments + graphical annotations]. Figure 6 shows the

critique presented with these methods. In this case, the *Critic Presenter* activates two components: the Text Critic and the Graphic Critic. The Text Critic presents the stored intro/reminder, demonstration, and evaluation messages from the violated constraint. The Graphic Critic executes function calls using the relevant furniture parts as parameters to annotate the designer's diagram.

## 7. Conclusion

Unfortunately, the field of design lacks a clear understanding of design critiquing, specifically, how to make a decision about which critiquing methods are appropriate in a certain critiquing situation. To advance knowledge in design critiquing as well as computer-based systems, we have developed a computational model of design critiquing using the Furniture Design Critic program. As a research tool this program provides a way to investigate design critiquing, specifically, the selection mechanisms. The example selection mechanism presented here is a plausible way to select delivery types and communication modalities.

The program makes an inference about a designer such as knowledge, weaknesses, and effective critiquing methods. It also records the history of *States* including what critiquing methods have been selected and if the feedback has been successful. It then selects a particular set of delivery types and critiquing methods by considering the inference results and the history of *States*.

Furniture Design Critic applies the constraint-based approach (Mitrovic *et al.*, 2007; Ohlsson, 1994) to a design domain. The *User Model* and the *Pedagogical Module* of constraint-based tutors are suitable to represent critiquing conditions and methods, and to articulate and explore the selection mechanisms of critiquing methods.

Our work contributes to the fields of design, design education, and computer-based design systems. The computational model presented here can be a foundation for us to describe a variety of critiquing strategies using the same factors, critiquing conditions and methods. For example, studio teachers can systemically explain and articulate their own critiquing strategies. Further, we as critiquing researchers now can perform empirical studies about appropriate applications of critiquing methods under specific conditions, because we have a systematic framework to describe critiquing strategies. In other words, this computational model can help us generate a set of variables that could be used to describe critiquing strategies and hypotheses that could be proved and tested. Our program, Furniture Design Critic also has closed a gap between what existing computer-based systems offer to design students and what studio teachers do.

We chose flat-pack furniture designing as an example domain. We plan to apply our model to other design domains such as architecture, product design, or engineering. The system mechanism that selects particular critiquing methods would be the same, because the system determines critiquing methods by considering domain independent information such as how many constraints are violated and satisfied and delivery types and modalities previously used. Also, it is not hard to add and

revise *Design Constraints* that represent domain knowledge and are used to identify critiquing opportunities and *Critiquing Rules* that determine the selection mechanism of the program.

## References


Aleven, V., Ashley, K., Lynch, C., & Pinkwart, N. (2007). *Proc. workshop on AIED applications for ill-defined domains*. The 13th International Conference on Artificial Intelligence in Education, Los Angeles, CA.

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

Anthony, K. H. (1991). *Design juries on trial: The renaissance of the design studio*. New York: Van Nostrand Reinhold.

Bailey, R. O. N. (2004). *The digital design coach: Enhancing design conversations in architecture education.* PhD Dissertation, Victoria University of Wellington.

Boyer, E. L., & Mitgang, L. D. (1996). *Building community: A new future for architecture education and practice*: The Carnegie Foundation for the Advancement of Teaching.

Chun, H. W., & Ming-Kit Lai, E. (1997). Intelligent critic system for architectural design. *IEEE transactions on knowledge and data engineering*, *9*(4), 625–639.

CORENET. (2009). CORENET e-Plan Check System. 2010, from http://www.corenet.gov.sg/corenet/

Fischer, G., Lemke, A. C., Mastaglio, T., & Morch, A. I. (1991). The role of critiquing in cooperative problem solving. *ACM Transactions on Information Systems, 9*(2), 123–151.

Fischer, G., McCall, R., & Morch, A. I. (1989). *Design environments for constructive and argumentative design*. The Human Factors in Computing Systems (CHI '89), Austin, Texas (p. 269–275).

Fu, M. C., Hayes, C. C., & East, E. W. (1997). SEDAR: expert critiquing system for flat and low-slope roof design and review. *Journal of Computing in Civil Engineering, 11*(1), 60–68.

Gertner, A. S., & Webber, B. L. (1998). TraumaTIQ: online decision support for trauma management. *IEEE Intelligent Systems*, *13*(1), 32–39.

Goldschmidt, G. (2002). *One-on-one: A pedagogic base for design instruction in the studio*. The Common Ground Design Research Society International Conference Brunel University (p. 430–437).

Graesser, A. C., Chipman, P., Haynes, B. C., & Olney, A. (2005). AutoTutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions in Education, 48*, 612–618.

ICTG (2009). The Intelligent Computer Tutoring Group (ICTG). Retrieved Aug, 20, 2009, from http://ictg.canterbury.ac.nz/

Koedinger, K. R., & Anderson, J. R. (1997). Intelligent tutoring goes to the big city. *International Journal of Artificial Intelligence in Education, 8*, 30–43.

Milik, N., Marshall, M., & Mitrovic, A. (2006). Responding to free-form student questions in ERM-tutor. *Lecture Notes in Computer Science, 4053*, 707–709.

Mitrovic, A. (2002). *Normit: A web-enabled tutor for database normalization*. The *International Conference on Computers in Education (ICCE)* (p. 1276–1280).

Mitrovic, A., Martin, B., & Suraweera, P. (2007). Intelligent tutors for all: The constraint-based approach. *IEEE Intelligent Systems*, *22*(4), 38–45.

Mitrovic, A., & Weerasinghe, A. (2009). *Revisiting ill-definedness and the consequences for ITSs. The 14th Conference on Artificial Intelligence in Education*, Brighton (p. 375–382).

Mostow, J., Aist, G., Burkhead, P., Corbett, A., Cuneo, A., Eitelman, S., *et al.* (2003). Evaluation of an automated reading tutor that listens: Comparison to human tutoring and classroom instruction. *Journal of Educational Computing Research*, *29*(1), 61–117.

Nakakoji, K., Yamamoto, Y., Suzuki, T., Takada, S., & Gross, M. D. (1998). From critiquing to representational talkback: computer support for revealing features in design. *Knowledge-Based Systems*, *11*(7–8), 457–468.

Oh, Y., Do, E. Y.-L., & Gross, M. D. (2004). *Intelligent critiquing of design sketches. The AAAI (American Association for Artificial Intelligence) Fall Symposium — Making Pen-based Interaction Intelligent and Natural* (p. 127–133), Washington DC.

Oh, Y., Gross, M. D., & Do, E. Y.-L. (2008). *Computer-aided Critiquing Systems: Lessons Learned and New Research Directions. Paper presented at the Computer Aided Architectural Design and Research in Asia (CAADRIA)* (p. 161–167).

Ohlsson, S. (1994). Constraint-based Student Modeling. In J. E. Greer & G. I. McCalla (Eds.), *Student modelling: The key to individualized knowledge-bases instruction* (pp. 167–189), Berlin: Springer-Verlag.

Ohlsson, S. (1996). Learning from Performance Errors. *Psychological Review*, *3*(2), 241–262.

Pinkwart, N., Aleven, V., Ashley, K., & Lynch, C. (2007). *Evaluating legal argument instruction with graphical representations using LARGO. The 13th International Conference on Artificial Intelligence in Education* (p. 100–108), Amsterdam, The Netherlands.

Robbins, J. E. (1998). *Design critiquing systems* (No. UCI-98-41): Department of Information and Computer Science, University of California, Irvine.

Robbins, J. E., & Redmiles, D. F. (1998). Software architecture critics in the argo design environment. *Knowledge-Based Systems*, *11*(1), 47–60.

Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books Inc.

Schön, D. A. (1985). *The Design Studio*. London: RIBA.

Simon, H. (1969). *The Science of the Artificial*. Cambridge: MIT Press.

Solibri.Inc. (2010, 09/13/2007). Solibri Inc. The World Leader in Design Spell Checking. Retrieved 07/12/2010, 2010, from http://www.solibri.com/

Suraweera, P., & Mitrovic, A. (2002). Kermit: A constraint-based tutor for database modeling. *Intelligent Tutoring Systems*, *2363*, 377–387.

Uluoglu, B. (2000). Design knowledge communicated in studio critiques *Design Studies*, *21*(1), 33–58.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., *et al.* (2005). *The Andes physics tutoring system: Five years of evaluations. The 12th International Conference of Artificial Intelligence in Education* (p. 678–685).

Wampler, J. (2002). Architecture Studio: Building in Landscapes. Retrieved Jan. 2009, from MIT Open Courseware: http://ocw.mit.edu/OcwWeb/Architecture/4-125Architecture-Studio–Building-in-LandscapesFall2002/CourseHome/index.htm

Weaver, N., O'Reilly, D., & Caddick, M. (2000). Preparation and support of part-time teachers: Designing a tutor training programme fit for architects. In D. Nicol & S. Pilling (Eds.), *Changing architectural education: Towards a new professionalism* (pp. 265–273), New York: Taylor & Francis Spon Press.

Zakharov, K., Ohlsson, S., & Mitrovic, A. (2005). *A feedback micro-engineering in EER-tutor. The Artificial Intelligence in Education (AIED)* (p. 718–725), Amsterdam, The Netherlands.