Design-First ITS Instructor Tool

The Instructor Tool allows instructors to enter problems into Design-First ITS through a process that creates a solution for a textual problem description and allows for revision and annotation by the instructor. The end result is a solution description that is used by the Expert Evaluator module of the Design-First ITS to evaluate student work and to provide input to the Student Model and the Pedagogical Agent. The Instructor Tool provides the benefit of automatically-generated solution suggestions that follow basic object-oriented design rules used by students, while also allowing for instructor variation and interpretation. Additionally, it is a learning tool for the instructor. Its generated solutions provide an evaluation of the clarity and completeness of the problem description, and offer insight into possible solutions which may not have been apparent to the instructor. Indeed, the latter has been my own experience with the tool; some of the solutions it generated initially surprised me and inspired new ways of thinking about design.

This document explains how to use the Instructor Tool by presenting guidelines on writing an appropriate problem description, then by stepping through two examples.

Problem Description Guidelines

Design-First ITS is intended to teach beginners how to identify basic design elements. It accurately generates solutions for problem descriptions that are appropriately structured for human students. Thus, the same requirements an instructor should follow in creating descriptions intended solely for human students apply to the Instructor Tool.

First, a good problem description for beginners must provide all the details necessary for a complete object-oriented design consisting of classes and their attributes

and methods. An explicit listing of all requirements clarifies the instructor's expectations and acknowledges that all students cannot be assumed to possess knowledge about any given problem domain.

Second, a good description does not provide details beyond those required. While developers in the real world must contend with identifying what data is pertinent to a problem, a student learns this skill through experience. Too much information early in a student's learning process can cause frustration and detract from the basic learning goals.

Third, problem details relating to program implementation (coding) should be excluded. The Design-First curriculum emphasizes solution design *before* coding, and Design-First ITS teaches object-oriented design. The focus is on high-level functionality rather than step-by-step details. Distinguishing between tasks that require a method versus a single line of code is a necessary skill that requires knowledge of procedural programming, and is improved primarily through experience. The Instructor Tool will generate superfluous methods if coding details are part of the problem description. The instructor can either revise the description and generate a new solution, or annotate them using the Instructor Tool, which would provide appropriate feedback (and a learning opportunity) to a student who makes the same error.

Finally, use simple declarative sentences, with basic subject/verb/object structure and active voice. Clarity rather than literary style is the goal. Long sentences with multiple clauses, use of passive voice and other complex grammatical forms can be confusing to both human students and to the Instructor Tool.

MontyLingua is the natural language processor used by the solution generation software within the Instructor Tool. It tags parts of speech, provides semantic

interpretation of names, dates and other proper nouns, and parses sentences into (verb, subject, object) tuples. It works best with simple sentence structure of the form subject, verb, object. It does, however, have the following limitations:

1. Appositives are not recognized: "An automated teller machine, or ATM, dispenses money." Do not include "or ATM." In many cases, specifying synonyms is not necessary because they are added automatically by the Instructor Tool. If a desired synonym is not included, it can be added manually.
2. Compound sentences are only partially processed: "The ticket machine prints tickets and returns the customer's change." This should be broken into two sentences.
3. Dependent clauses are not recognized: "If the tickets are not sold out, the machine prints tickets for the customer." This should be split into two sentences. Another example is "The machine counts the number of tickets sold." "…tickets sold" implies "tickets that are sold," a dependent clause that describes tickets. This also should be reworded to avoid more than one verb in the sentence.

In summary, follow these grammatical rules in writing problem descriptions:

1. Use declarative sentences with simple subject/verb/object structure.
2. Use the active voice ("The ATM dispenses money.") rather than the passive voice ("Money is dispensed by the ATM.").
3. Add synonyms through the Instructor Tool rather than as appositives in the problem text.
4. Write out numbers as words ("three") rather than numerals ("3").

The problem descriptions in the two examples that follow illustrate the appropriate style. While it may at first seem overly simplistic, its meaning is clear to the human reader. Future enhancements to the MontyLingua software may allow more grammatical variety and a more natural style.

Entering Problems into Design-First ITS

Example 1: Movie Ticket Machine

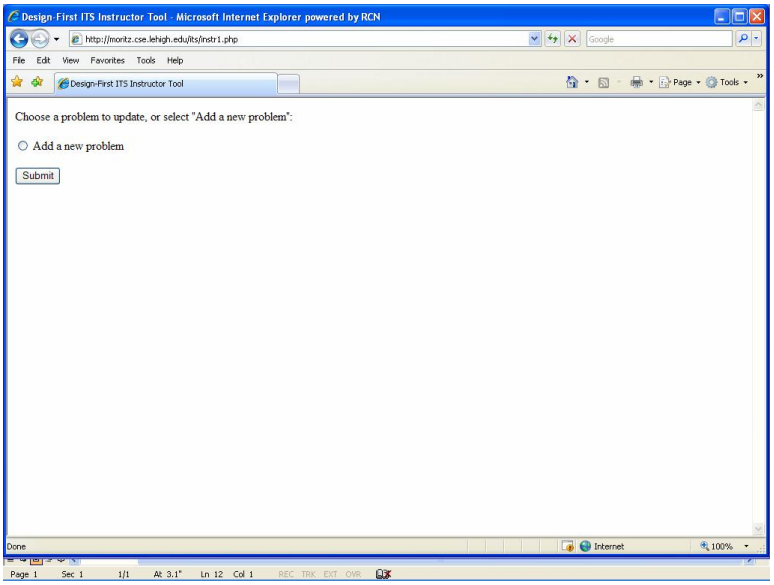


Figure 1: Initial Instructor Tool Screen

1. The initial Instructor Tool screen presents a list of existing problems in the database, plus an option to add a new problem. Click on "Add a new problem" and press "Submit."

2. Enter the title and description in the fields provided. The user interface option will generate the classes needed for either a character-based or graphical interface.

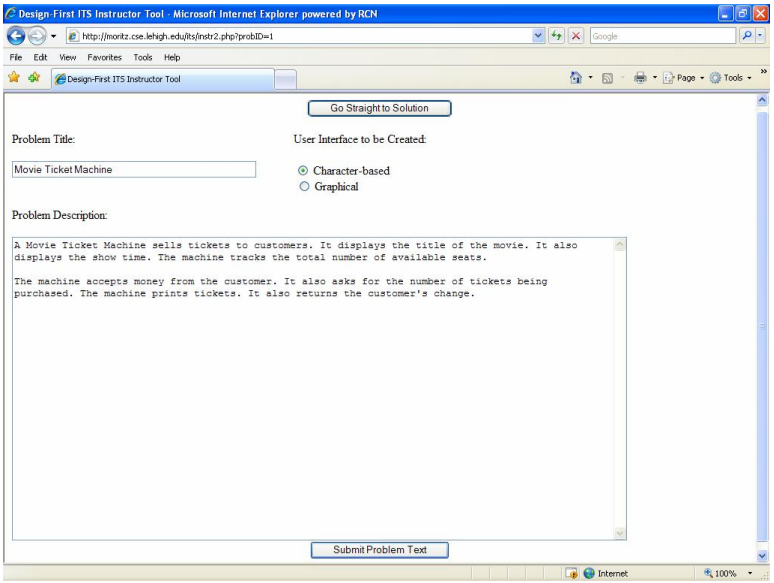


Figure 2: Problem Description Entry

The problem description is:

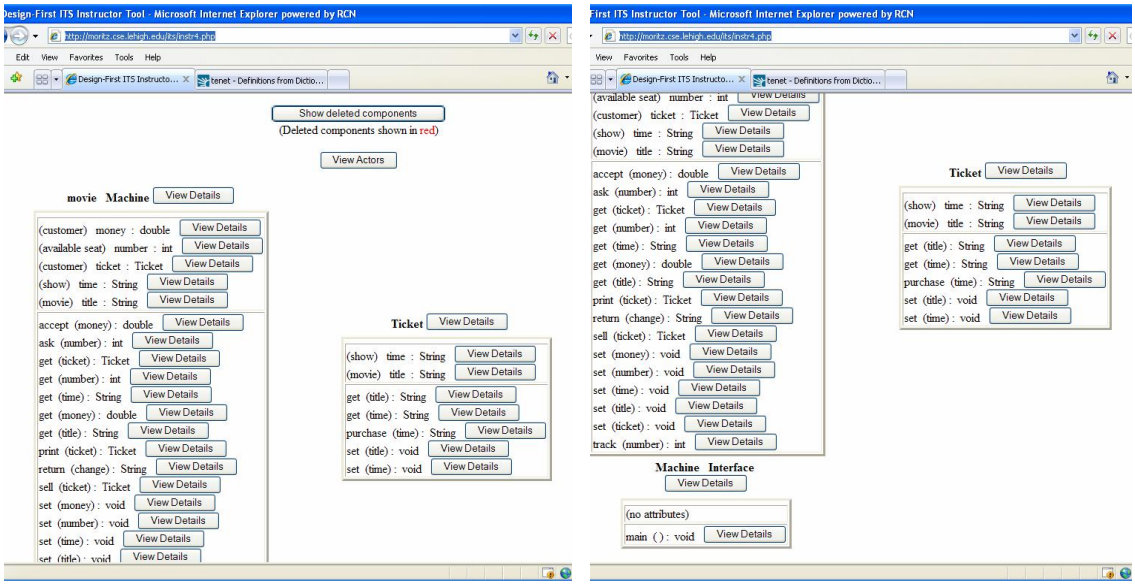
A Movie Ticket Machine sells tickets to customers. It displays the title of the movie. It also displays the show time. The machine tracks the total number of available seats.

The machine accepts money from the customer. It also asks for the number of tickets being purchased. The machine prints tickets. It also returns the customer's change.

Notice that the problem description is concise. It includes the necessary functions of a machine that sells movie tickets and nothing else. It contains simple sentences, all in the active voice.

3. Press the "Submit Problem Text" button. This deletes any existing solution components for this problem, and generates a solution from the text. If the process completes within 30 seconds, a class diagram is displayed. However, the analysis usually takes longer, in which case a message is displayed asking the user to view the result later.

4. To view the class diagram, go to the problem description screen and click on "Go Straight to Solution". Do not click on the "Submit" button unless you want to start from scratch; it will delete all instructor annotations as well as the existing solution.



Figures 3a, 3b: Generated solution to Movie Ticket Machine

5. The instructor can view the details about any class, attribute or method by clicking on the "View Details" button next to the component name. Before proceeding to view and change components, let us examine the design that was generated.

Three classes have been created: Machine, Ticket and Interface. The Interface class was generated for the character-based user interface option that was selected; the other two classes come from nouns found in the text. These nouns are either subjects of sentences, or were objects that were found to have some characteristics of their own and were not identified as simple data items. If a noun was used with adjective(s), one of those adjectives appears in parentheses before the class name.

6. Click on "View Details" next to (movie) Machine. The following screen appears:

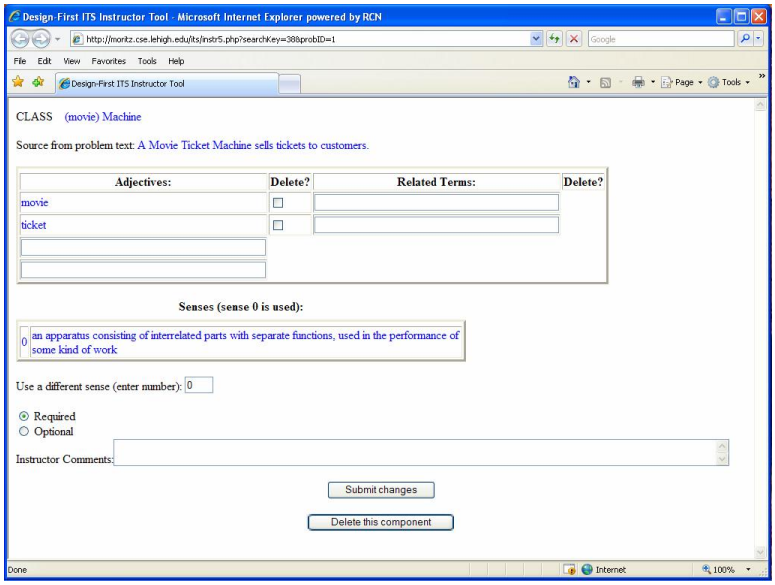


Figure 4: Details of Machine class

Fields on this screen are:

- Source from problem text - This is the first sentence from which this component was derived.
- Adjectives - These are all adjectives for the component found in the problem description. Adjectives may be added by typing into the blue-outlined fields; they may be deleted by checking the box next to the name.
- Related terms - Synonyms for the class name. They come from WordNet, which is also the source of "Senses." Related terms may also be added and deleted manually.
- Senses – "Sense" is WordNet's term for definition. Many words have multiple senses, listed in order of frequency of use. The Instructor Tool chooses the first sense (number 0) by default, but the instructor may choose a different sense. Doing so invokes the Instructor Tool to replace the related terms based on the new definition.
- Required/Optional - Required is the default for all components except get and set methods. The instructor may mark any component optional, or change an optional component to required.
- Instructor Comments – Comments are used to provide additional feedback to the student explaining the purpose of the component, or why it may be

optional. Comments are available to the Pedagogical Agent for display in the student interface of Design-First ITS.

- "Submit changes" button – Any changes entered in the above fields will be processed when this button is pressed.
- "Delete this component" button – Used to mark the component deleted. The instructor must enter a reason for deleting a component; it is entered on a separate screen that is displayed automatically.

7. Close the Machine detail screen and return to the class diagram. We will now delete the Ticket class. Click on "View Details" next to the Ticket class name.

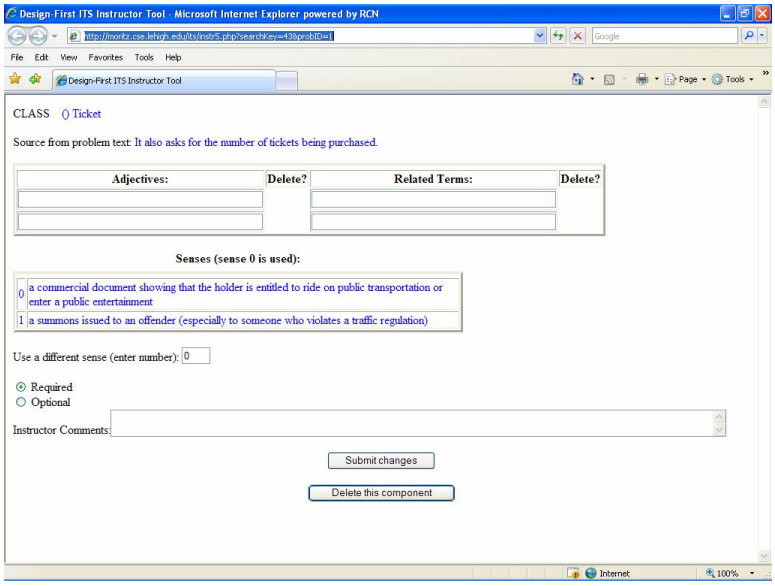


Figure 5: Details of Ticket class

8. Click on "Delete this component" and enter "*Although ticket is an object in the problem, it is only printed as needed, not saved between customer transactions. It is not a "persistent" object in the problem*." into the Instructor Comments field on the next screen, then click "Submit." You will receive a confirmation of the delete.

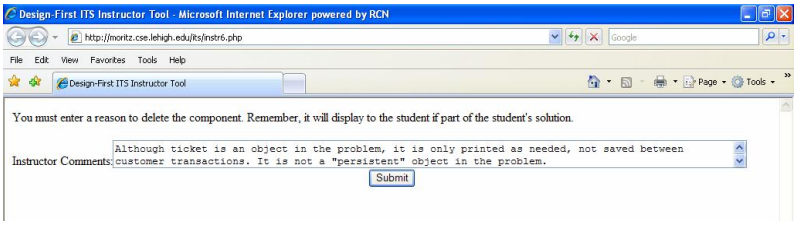


Figure 6: Reason for deleting class

9. Back on the class diagram, click on "Show deleted components" at the top. The screen will be redisplayed, with deleted components in red. All components of the class have been marked deleted.

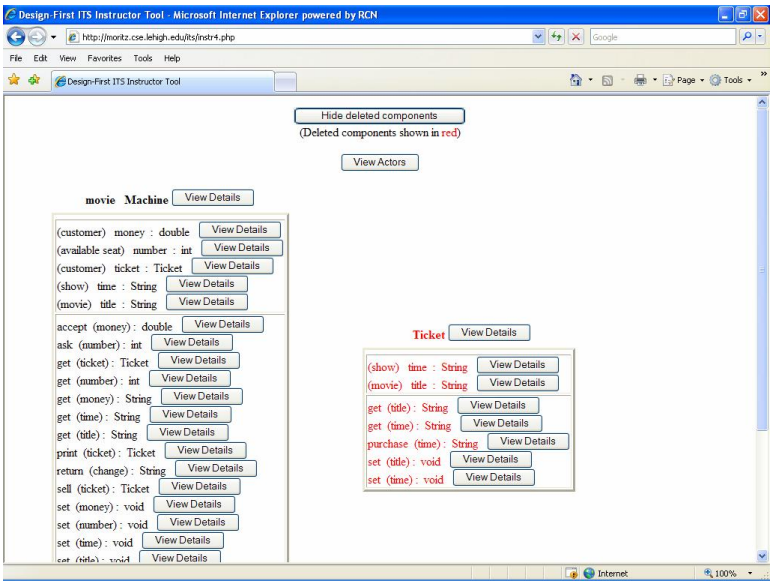


Figure 7: Class diagram with deleted components

10. Notice that there is an attribute "ticket" of type "Ticket" in the Machine class. Delete this attribute through the same process: click on "View Details," "Delete this component" and enter a reason. The attribute and its get and set methods will be marked deleted.

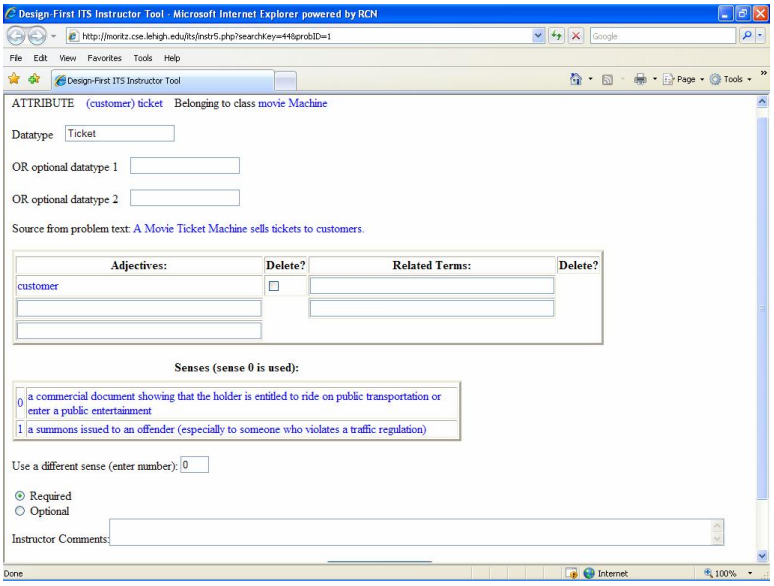
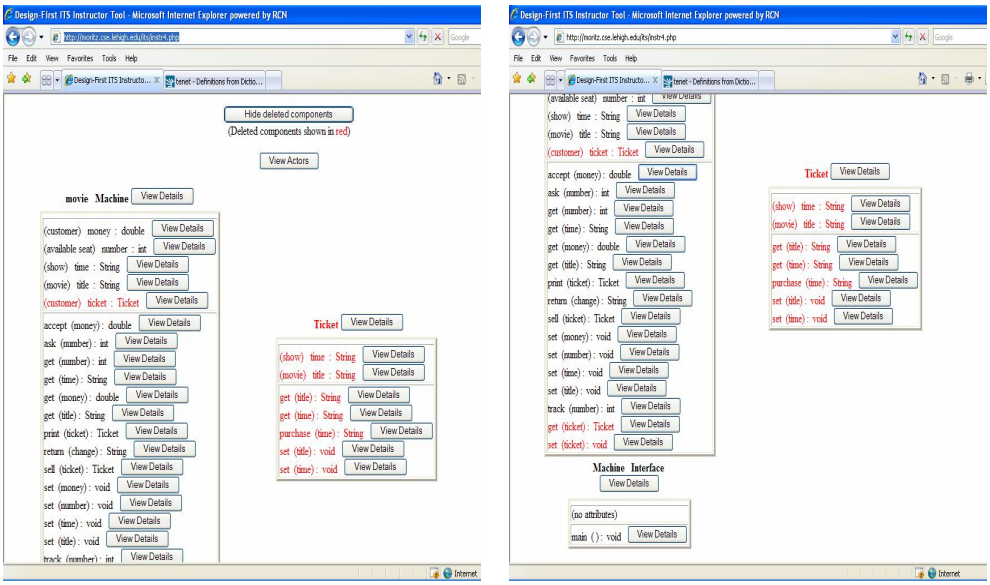


Figure 8: Details of attribute ticket

The attribute detail screen looks like the class detail screen, except that the datatype is shown. The user may change the automatically-selected datatype, and may also add up to two alternate acceptable datatypes.

11. The ticket attribute and its get/set methods are shown after deletion.



Figures 9a, 9b: Result of deleting attribute ticket

12. There are two methods remaining related to the deleted class Ticket: print (ticket) and sell (ticket). We view the details of print (ticket) first.

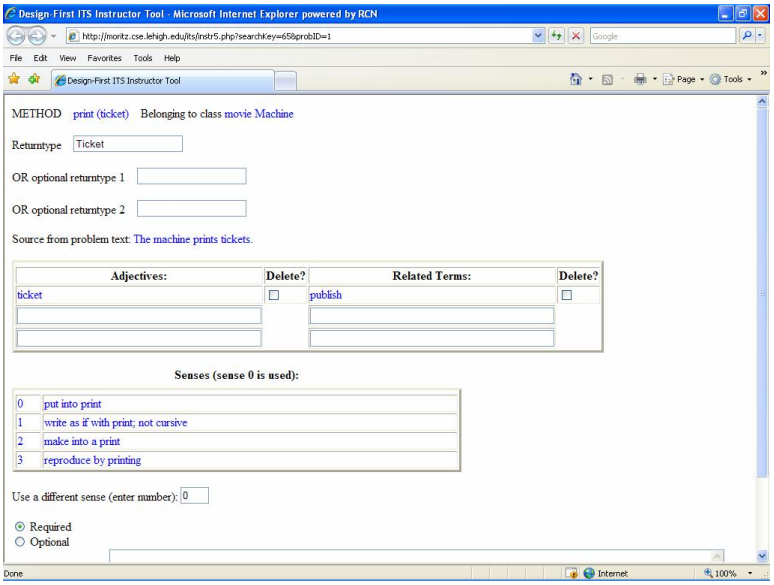


Figure 10: Detail of method print (ticket)

The method detail screen is similar to the attribute detail screen. Instead of datatype, the returntype of the method is shown. Alternate returntypes may be entered. The adjectives listed for a method are really the direct object(s) of the verb (which is the method name). This print method prints tickets. If more than one adjective were displayed, they would all be synonyms of the word ticket.

Change the Returntype from Ticket to void. Also enter boolean as an optional return type. Click "Submit" to save the changes. A screen verifying the update is displayed; click the button to redisplay the updated method.

13. We now address the method "sell (ticket)." The problem text from which this method came refers to the entire process. It does not represent a step or a discrete function the Machine performs. Delete this method using the same process by which an attribute is deleted.

14. Figure 11 shows the class diagram after our changes (deleted components are hidden). The instructor should review the details of every remaining component to make sure the correct sense has been chosen, all acceptable data/return types are correct, all adjectives and related terms have been entered, and each is correctly marked as required or optional. Additional instructor comments should be added to enrich the feedback provided to students.

    The next few steps illustrate some common changes.

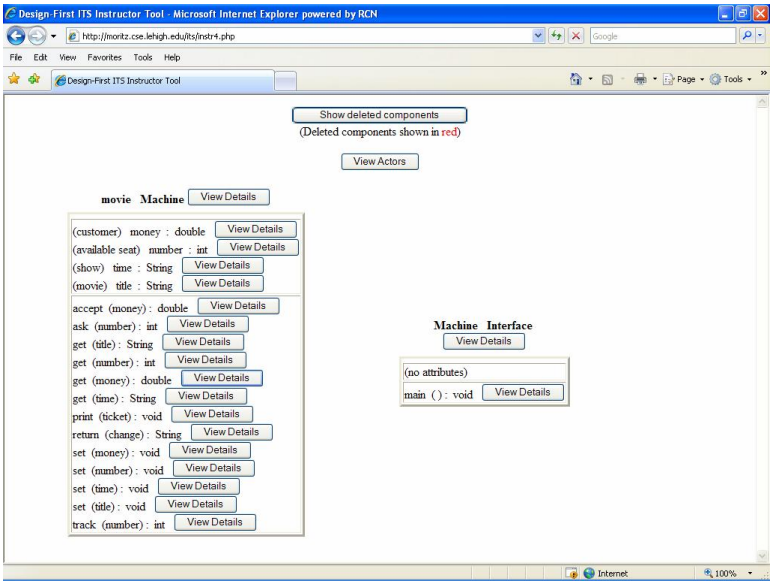
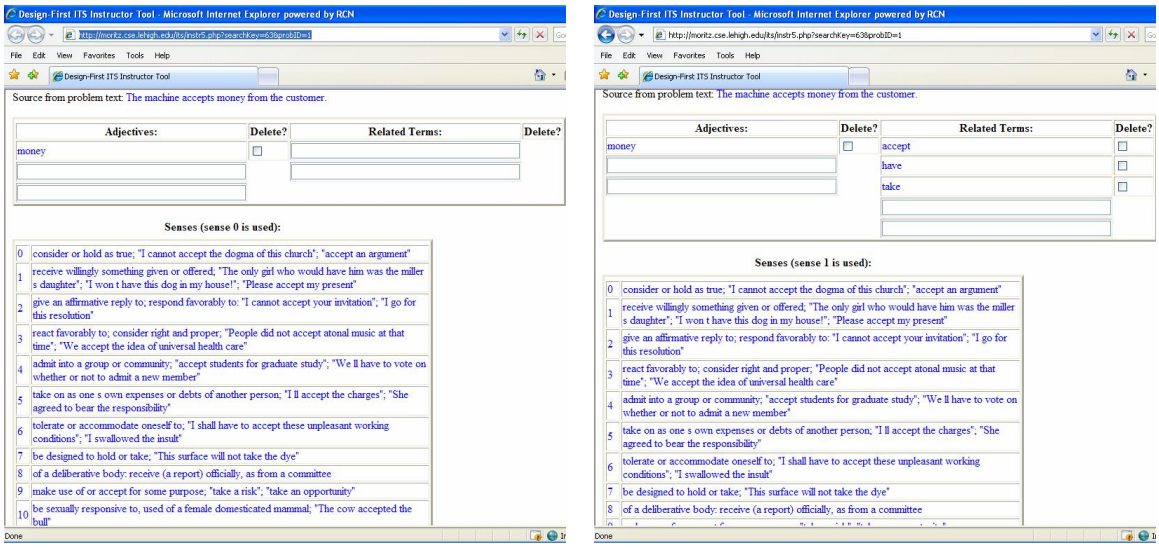


Figure 11: Class diagram after changes

15. View details for accept (money). The sense that was chosen to define accept is not appropriate. The second choice, "*receive willingly something given or offered*," is a better meaning for the problem context. Enter its sense number (1) in the "Use a different sense" field. A portion of the original and resulting details screen for accept (money) is shown in Figures 12a and 12b. Notice how new related terms appropriate to the new definition have been added.

16. Method track (number) was added for tracking the number of seats left in the theater. This doesn't really need its own method – it could be accomplished with a single line of code. An instructor could choose to delete it, or accept it as appropriate in some designs (perhaps as a method that returns the number of tickets remaining). Figure 13 shows the revisions an instructor could enter to support it as an optional component.



Figures 12a, 12b: Before and after changing sense of method accept (money)

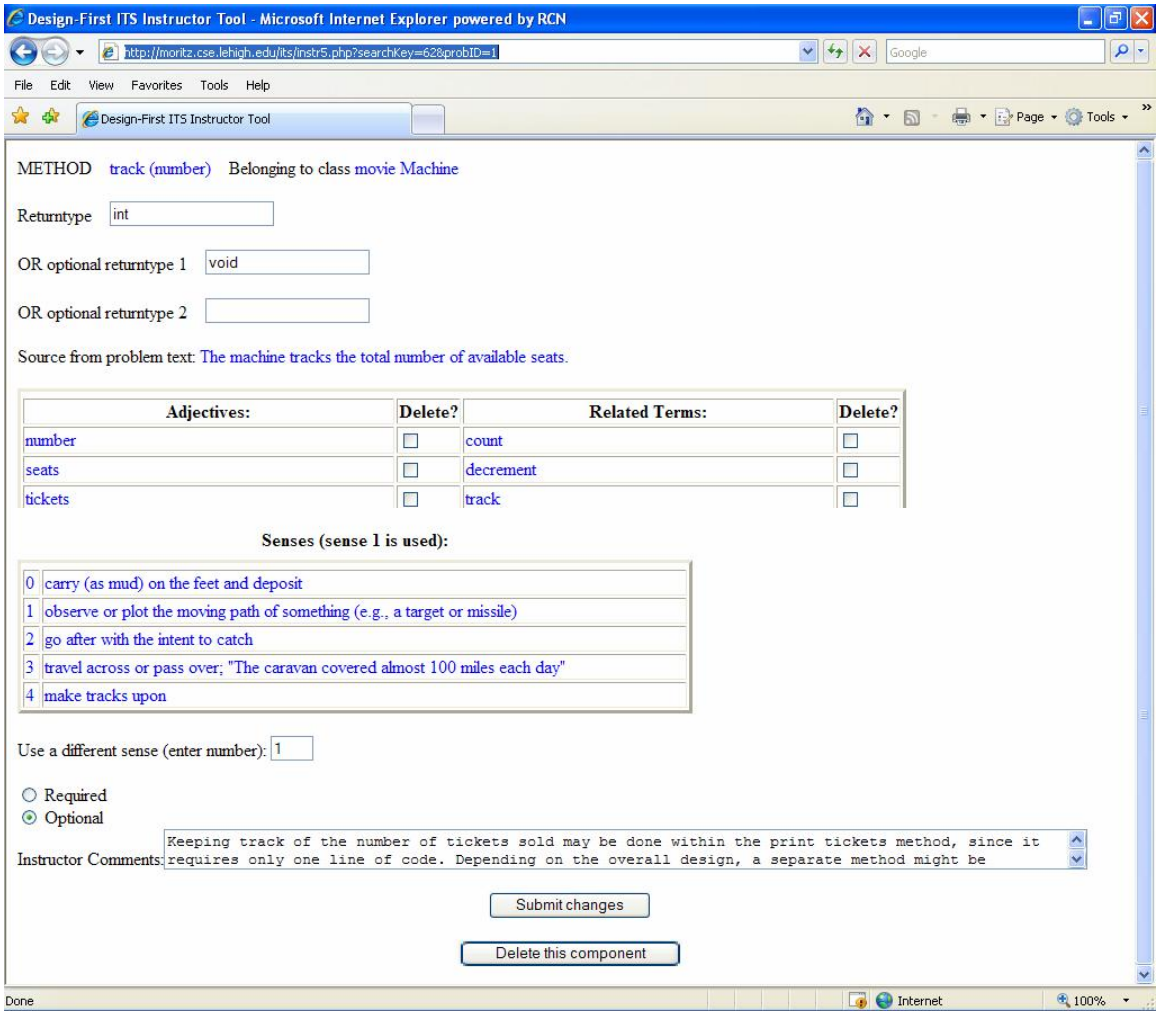


Figure 13: Method track (number) after revisions

17. The "View Actors" button on the class diagram screen displays a list of all the actors extracted from the problem description. The "Update/Delete" button next to the actor allows entry of instructor comments or deletion of the actor. Deletion requires a comment, just as for all other components.

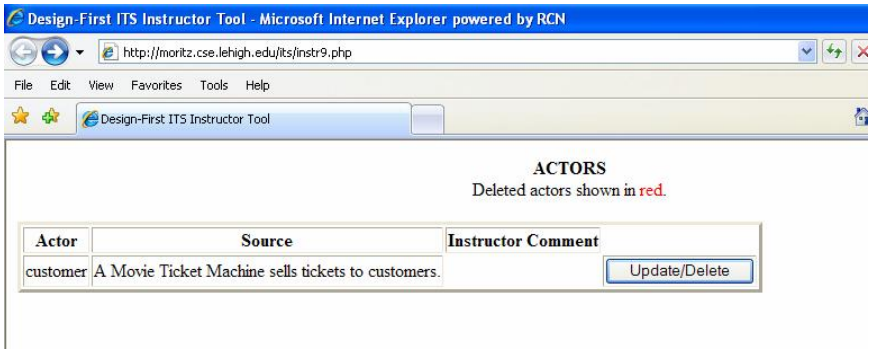


Figure 14: Actors screen

This completes Example 1. By following the steps outlined for all solution components, a complete solution with optional variations was built. All deleted components include an explanation of why they are not suitable for a correct solution. All variations allowed through optional components are identified by the instructor, allowing the instructor's judgment to determine what Design-First ITS evaluates as acceptable. Instructor comments allow the instructor's own thoughts to be conveyed to the student by the ITS' Pedagogical Agent.

A notable omission in the Instructor Tool is the ability to add components directly to the solution. One of the principles of Design-First ITS' problem solving approach is that all required knowledge for a problem be explicitly stated in the problem description. If a desired component does not appear in the automatically-generated solution, the problem text should be revised to include that component and a description of its role in the overall problem. A new solution may then be generated from the text.

Example 2: Automated Teller Machine

Example 2 is included to give the user a deeper sense of how the Instructor Tool interprets problem descriptions. An increasing familiarity with this aspect of the tool will improve the instructor's ability to create descriptions that are more concise and accurate, and yield good generated solutions. We also show a process which the instructor can follow in evaluating and revising a generated solution.

1. Select "Add a new problem" on the initial Instructor Tool screen, then enter "ATM Machine" for the problem name and the following text for the problem description:

   An ATM dispenses money from an internal supply. It reads a customer's card number. It also reads the customer's password. It verifies that the card number and password are correct. The ATM contacts the bank's central database for the verification and customer balance.

   The ATM can display the customer's balance. It can deposit money into the customer's account. It also withdraws money from the account. The ATM prints a receipt showing the type of transaction and account number. The receipt also shows the new account balance.

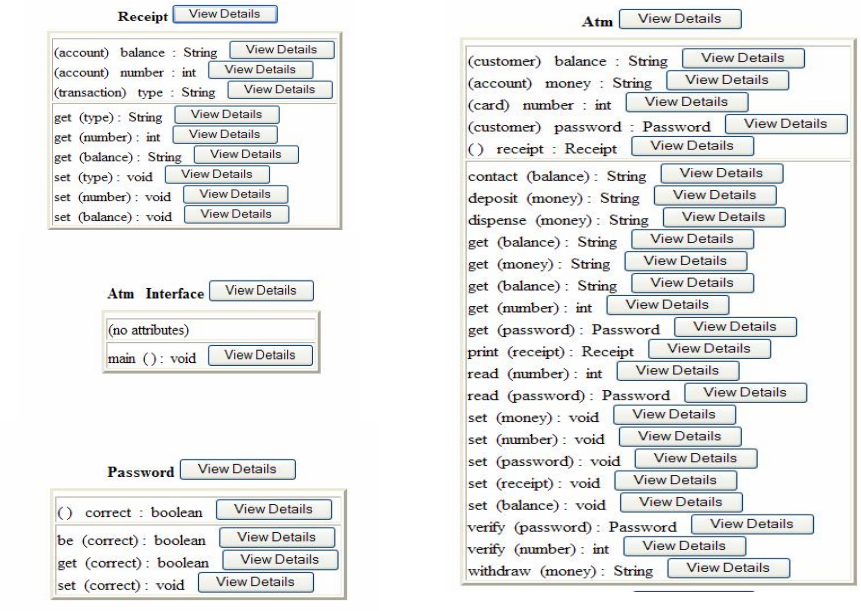
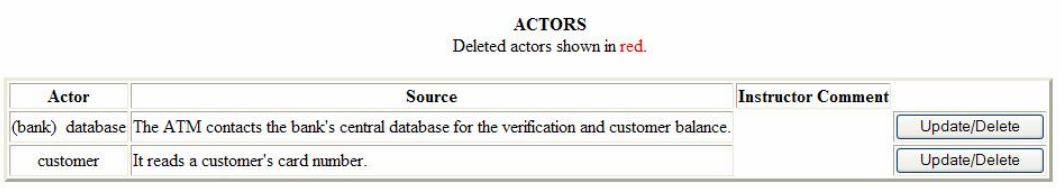


Figure 15: Class diagram generated for ATM problem



ACTORS

Deleted actors shown in red.

Actor	Source	Instructor Comment	
(bank) database	The ATM contacts the bank's central database for the verification and customer balance.		Update/Delete
customer	It reads a customer's card number.		Update/Delete

Figure 16: Actors generated for ATM problem

2. Classes should be examined first.
    a. The ATM class is required.
    b. A separate class for Receipt is questionable. We could treat it like the Ticket class in the Ticket Machine problem, by considering it a short-lived object that is printed by a print receipt method. But a valid design could also treat it as a class, and create a Receipt object when one needs to be printed. This would also require moving the print (receipt) method to the Receipt class. Copying a method from one class to another is not currently supported in the tool, but may be recommended as a future enhancement. Even so, the instructor may mark this class as optional, with an explanation in the comments field.
    c. Password should not be a class. Typically, a password is just a character string that does not have any behaviors.
    d. The Atm Interface class is needed as a driver and to control the user interface.

    Make note of which classes need revisions or deletion, but don't enter the changes until after completing step 3.

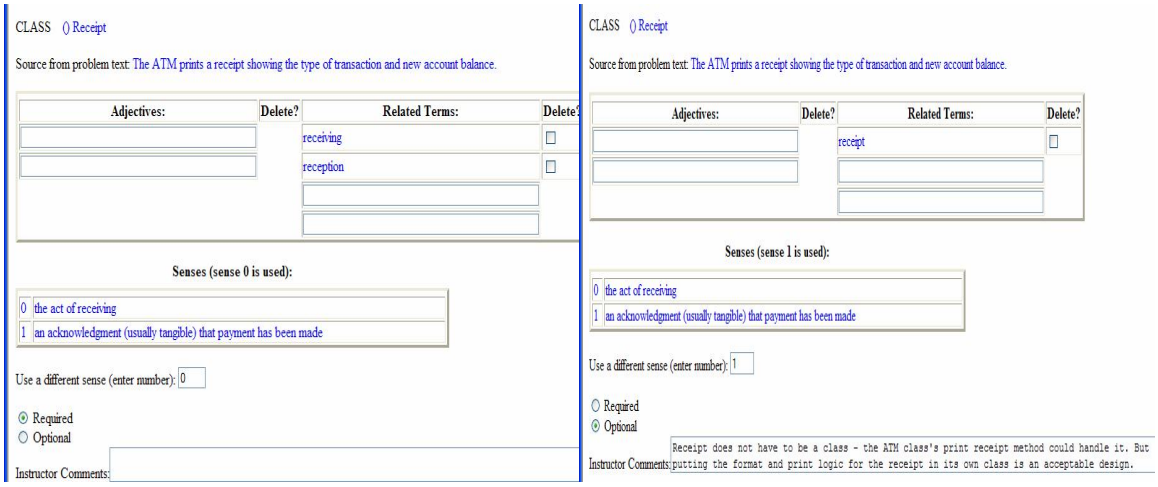
3. Look for missing classes. If an expected class is not present, review the problem description. Remember that every component of the problem must be mentioned in the text, along with a description of actions and data that it includes. Attributes and methods will also not be part of the design if they are not included in the description.

    Missing attributes and methods in the classes that are a valid part of the solution should also be considered at this time, since revising the problem description requires regeneration of the entire solution and wipes out annotations already made to other components.

    If a missing component is included in the description, try rewording the text that pertains to it. Were all sentences simple subject/verb/object structures, in the active voice, with no appositives? Split multiple clauses into separate sentences.

    Text revision and solution regeneration is expected to be an iterative process. Experience with the tool will result in more thorough initial descriptions and fewer iterations needed to create a complete solution, as well as fewer misunderstandings by students.

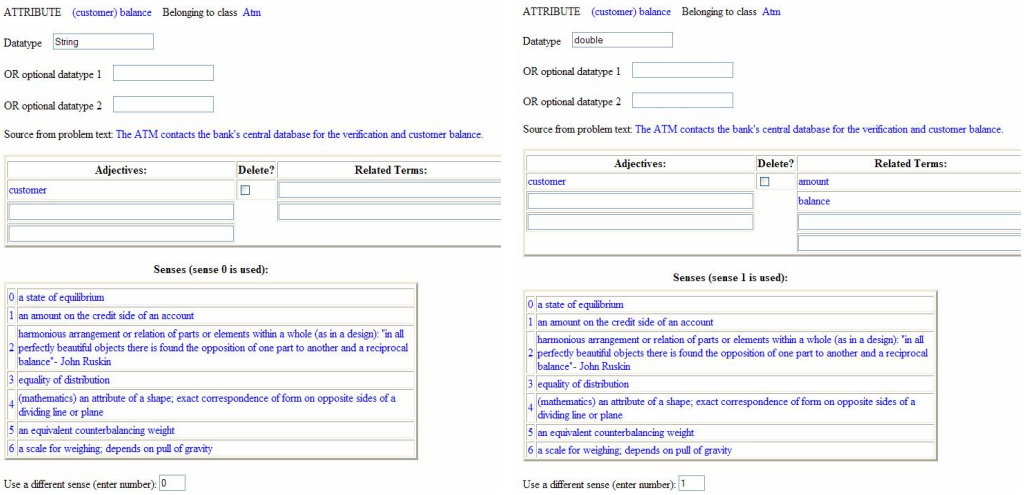
4. After text revisions produce all expected components, enter the changes to the classes noted in step 2. Also review all classes generated as a result of the revisions to the description.



Figures 17a,17b: Receipt class before and after revision

5. Now carefully review the attributes and methods of all valid classes. We describe changes to only a few of the components of the ATM class, to illustrate some common situations.

   - (customer) balance – Notice the datatype is incorrectly set to String. On viewing the details for the attribute, we see that an inappropriate sense was chosen as the definition. We change the sense to "*an amount on the credit side of an account.*" The datatype is automatically corrected to double, and related terms for the new definition are added.



Figures 18a,18b: (customer) balance before and after revision

   - (account) money – The source of this attribute is "*An ATM dispenses money from an internal supply.*" That means this attribute keeps track of the amount

of money in the ATM for distribution to customers. *Account* is not a suitable adjective for this attribute. We delete it and adjective *customer account. Internal* is retained as a valid adjective, and we add *atm* and *machine* as adjectives. We also add related terms *balance* and *cash*, and correct the datatype (it was String, but should be double).

- (customer) password – Should be deleted, not because the Password class was deleted, but because there is no need to store the password. It is needed only to verify that it is correct, which is handled in a single method *verify (password)*. It can be passed to that method as a parameter.
- receipt – Also not needed as an attribute. Whether created in a Receipt class or the print receipt method, it exists only for the current transaction.
- contact (balance) – This is an example of an extraneous method created due to a flaw in MontyLingua's processing of the sentence containing this verb. It can be deleted with a notation that it is a solution generation error.
- read (password) – Reading data should be delegated to the user interface portion of a system. That allows for separation of logic and interface, and simplifies coding changes needed to replace the user interface. Additionally, password is not an attribute, so there is no "get" method needed. Thus, method should be deleted.
- verify (number) - This method is redundant with *verify (password)*. When the ATM checks the customer's password, it needs the card number as well. So verifying that the card number is valid is part of the process of *verify (password)*.

The solution will be complete when a review of every attribute and method in each class done, with changes and comment entered as required.

This example is intentionally more complicated than the first to illustrate the thought process by which an instructor constructs a problem for her students. The Instructor Tool was designed to aid this process. The intended results are clear, well-defined problems with no misunderstandings in student interpretation or surprises in valid solutions. Additionally, the Tool provides sufficient customizations and commenting of the generated solution to reflect the instructor's preferred design techniques and teaching style. The final solution set can be as restrictive or permissive of variation as the instructor desires.