Chapter 19
Authoring Problem-Solving Tutors: A Comparison between ASTUS and CTAT

Luc Paquette, Jean-François Lebeau, and André Mayers

Computer Science Department, Université de Sherbrooke,
2500 boulevard de l'Université, Sherbrooke, Québec, J1K 2R1, Canada
{luc.paquette,andre.mayers}@USherbrooke.ca

Abstract. ASTUS is an Intelligent Tutoring System (ITS) framework for problem-solving domains. In this chapter we present a study we performed to evaluate the strengths and weaknesses of ASTUS compared to the well-known Cognitive Tutor Authoring Tools (CTAT) framework. To challenge their capacity to handle a comprehensive model of a well-defined task, we built a multi-column subtraction tutor (model and interface) with each framework. We incorporated into the model various pedagogically relevant procedural errors taken from the literature, to see how each framework deals with complex situations where remedial help may be needed. We successfully encoded the model with both frameworks and found situations in which we consider ASTUS to surpass CTAT. Examples of these include: ambiguous steps, errors with multiple (possibly correct) steps, composite errors, and off-path steps. Selected scenarios in the multi-column subtraction domain are presented to illustrate that ASTUS can show a more sophisticated behavior in these situations. ASTUS achieves this by relying on an examinable hierarchical knowledge representation system and a domain-independent MVC-based approach to build the tutors' interface.

19.1 Introduction

Intelligent Tutoring Systems (ITS) that support a learning-by-doing pedagogical strategy are usually developed in line with one of three established approaches: model-tracing tutors (Anderson and Pelletier 1991), constraint-based tutors (Mitrovic et al. 2003) and example-tracing tutors (Aleven et al. in press; Razzaq et al. 2009). All of these fit VanLehn's tutoring framework (VanLehn 2006), in which a model of a task domain is used to evaluate each of the learner's steps (which are themselves driven by mental inferences) as correct or incorrect. Model-tracing tutors such as Cognitive Tutors (Anderson 1995) and Andes (VanLehn 2005) have been proven to be successful in the classroom (Koedinger et al. 1997), but their success is mitigated by their cost (Murray 2003), which is mainly due to the effort needed to develop a generative model of the task domain. Both constraint-based and example-tracing tutors are designed to reduce this cost. The former use an evaluative model involving constraints defined over a set of pedagogically relevant solutions, and the latter, a task-specific evaluative model built from a domain expert's interactions with the learning environment. Example-tracing tutor frameworks can offer tools to generalize the resulting model (Aleven et al. in press; Matsuda 2005), but adding such levels of complexity is an obstacle to their democratization, and still does not make them as flexible and comprehensive as model-tracing tutors. Constraint-based tutors may be particularly effective in handling ill-defined tasks in well-defined domains, such as design-based ones; however, they cannot follow learners as closely as model-tracing tutors, a capacity which is especially interesting for well-defined tasks. To reduce the effort needed to develop model-tracing tutors, one approach is to rely on a more (Cognitive Tutors) or less (Andes) domain-independent framework. In such a context, the knowledge representation system used to build the model is a key part of the framework, and its expressivity and reasoning capacity determine which domains can be modeled and how straightforward it is to model one.

Our work is based on the hypothesis that a more sophisticated knowledge representation system not only widens the range of domains that can be modeled, but also facilitates the testing of varied domain-independent pedagogical strategies, including some that are more elaborate than the ones usually found in model-tracing tutors. In fact, our efforts can be seen as an attempt to achieve an objective similar to that of Heffernan (Heffernan et al. 2008), who expanded Cognitive Tutors with a domain-specific (algebra) pedagogical model in order to provide tutorial dialogs closer to those used by experienced human tutors. Thus, our framework ASTUS is designed with two objectives: to reduce the prohibitive effort usually associated with the development of model-tracing tutors, and to provide the ITS community with a modular framework. In terms of Wenger's classic ITS architecture (Wenger 1987), ASTUS's domain-independent expert and interface modules interpret domain-specific models, and pedagogical and learner model modules can be customized to try out different pedagogical avenues. To achieve this, ASTUS uses a knowledge representation system that can be seen as a middle ground between the Cognitive Tutors production systems and the Andes solution graphs, in which the set of next possible steps is updated before each of the learner's actual steps. This approach, which is online (like Cognitive Tutors) but also top-down (like Andes), was adopted under the hypothesis that the advantages of both can be combined. Designed to model domains from a pedagogical perspective rather than to model the cognitive process used to solve them, ASTUS's knowledge representation system represents procedural knowledge hierarchically, using knowledge components of different grain sizes. Of these components, the examinable ones represent the skills explicitly tutored and the black-box ones model the underlying required abilities. Finally, to ensure that the tutor has complete access to the learning environment interface, as required by Anderson et al. (1995), these knowledge components act as a Model in terms of the Model-View-Controller (MVC) architectural design pattern on which the interface module is based.

In this chapter we present a study that evaluates the effectiveness of ASTUS by comparing it with Cognitive Tutor Authoring Tools (CTAT), a model-tracing tutor framework derived from the Cognitive Tutors (Koedinger 2003). In order to compare the two frameworks, we used each of them to author a tutor for the multi-column subtraction task domain. Each contains a generative model that can be used to trace a correct problem-solving path, but also many different incorrect ones. The objectives of this study are to evaluate whether both frameworks allow us to exhaustively model the multi-column subtraction domain; to identify the features that make each framework more or less suitable for the situations covered by a given set of scenarios; and to show which types of pedagogical behavior are made possible by each framework. The tutors were authored with the sole purpose of comparing the frameworks; we do not plan to experiment with them in a classroom.

In the next sections of this chapter, we describe the two ITS frameworks (CTAT in Section 2, ASTUS in Section 3). For each, we begin by giving an overview of its design and then present its knowledge representation system in detail, with examples from the multi-column subtraction domain. Section 4 presents the methodology of our study, detailing each step that was taken in order to achieve our objectives. For instance, we explain why we deemed the multi-column subtraction domain a good choice to evaluate the strengths and weaknesses of the two frameworks. In Section 5, we discuss the features of the resulting tutors and present scenarios which illustrate how each framework deals with different situations encountered in modeling the multi-column subtraction domain. In particular, we discuss the difference between the frameworks' respective tracing algorithms, their difficulties in modeling specific situations and the types of pedagogical behavior they support. Finally (Section 6), we conclude that authoring a tutor in either framework is a similar task, but that with ASTUS, more attention must be paid in building the task domain model so that it can be fully exploited by the domain-independent pedagogical module. However, without any extra domain-specific effort, the resulting tutor offers elaborate pedagogical behaviors that are usually not supported in problem-solving tutors. The chapter ends with a brief presentation of future work towards a new version of the ASTUS framework.

19.2 The CTAT Framework

CTAT[1] is a freely distributed domain-independent ITS framework that can be used to create tutors for various well-defined task domains (examples include stoichiometry and genetics). The main objective of CTAT is to reduce the amount of work required in authoring these tutors (Aleven et al. 2006). The CTAT framework allows the creation of two different types of tutor: Cognitive Tutors and Example-Tracing Tutors) (Aleven et al. 2006). Cognitive Tutors are based on the ACT-R theory of cognition (Anderson 1993; Anderson and Lebiere 1998) and use a cognitive model of the skills being tutored. To create such a model, expertise in AI programming is required, as the model may be implemented using the Cognitive Tutor Development Kit (TDK) (Anderson and Pelletier 1991) or the Jess rule

[1] http://ctat.pact.cs.cmu.edu

engine[2] (only the latter is distributed along with CTAT). In our study, the CTAT tutor was created as a Cognitive Tutor implemented via the Jess rule engine. In the following sections, when we speak of the CTAT framework, we are referring to Jess-based Cognitive Tutors authored using CTAT.

19.2.1 Knowledge Representation

CTAT tutors, being derived from the TDK-based Cognitive Tutors, use Jess production rules to represent procedural knowledge and Jess facts to represent declarative knowledge. These facts, referred to as Working Memory Elements (WMEs), can be used to represent the task, model the learner's perception of the interface and store temporary results in working memory. The content of a WME is defined by slots that can be filled with primitive data (boolean, integer, string, etc.) or references to other WMEs. For example, in our multi-column subtraction tutor, the WME for a column contains the following slots (the type of each slot is indicated here for clarity's sake; it is not specified in the actual model):

```
WME Column {                                WME Textfield {
  name [string]                               name [string]
  nextColumn [Column]                         value [string]
  previousColumn [Column]                   }
  minuend [Textfield]
  subtrahend [Textfield]                    WME DecrementColGoal {
  difference [Textfield]                      column [Column]
  hasBeenBorrowedFrom [boolean]     }
}
```

Production rules provide a cognitively plausible path to explain a learner's actions, whether they are steps in the interface or mental inferences. Thus, they can exhibit two kinds of behavior (on the RHS): recognizing a step (no more than one per rule) from an event sent by the interface, or updating the content of the working memory (adding, modifying or removing WMEs) to reflect a learner's mental inferences.

```
(defrule AddDrementColumnGoal
 ?problem <- (problem (subgoals $?first ?evaluateGoal $?rest)
 ?evalGoal <- (evalColumnDecrementationGoal (column ?col))
 ?col <- (column (minuend ?minuend &:(neq ?minuend 0))
=>
 (bind ?decColGoal (assert (decrementColGoal (column ?col))))
 (modify ?problem
(subgoals (create$ $?first $?rest ?decColGoal))))
```

The above rule is fired when there is a goal of evaluating the decrementation of a column's minuend for a column with a minuend different from zero. The WME it creates will allow other rules to match, in a chain, including the final one that will result in the action of decrementing the minuend. The following rule is triggered if there is a goal of subtracting the problem's current column, and the step is

[2] http://www.jessrules.com

recognized if the correct value is entered in the column's difference slot, as specified by the *predict-observable-action* statement.

```
(defrule Subtract
 ?problem <- (problem (currentColumn ?column)
 (subgoals $?first ?goal $?rest))
 ?goal <- (subtractColumnGoal (column ?column))
 ?column <- (column (difference ? diff)
 (minuend ?minuend)
 (subtrahend ?subtrahend))
 =>
 (bind ?difference (- ?minuend ?subtrahend))
 (predict-observable-action ?diff WRITE-VALUE difference))
```

When a step is executed in the interface, CTAT tries to find a chain of rules that leads to it. A loop is initiated in which the content of the currently available WMEs is compared with the rules' firing conditions in order to find matches. As rules can alter the content of the working memory when they are fired, additional production rules can be fired and the matching process is started over until the rule producing the learner's step is found. If no such rule is found, the step is considered an error. In CTAT, pedagogically relevant errors are modeled using production rules marked as "buggy". Buggy rules, like normal ones, can either match a step or modify the content of the working memory. In both cases, errors are detected after exactly one incorrect step, when the chain of production rules that leads to this step contains a buggy rule.

19.3 The ASTUS Framework

ASTUS, like CTAT, is designed to be a domain-independent ITS framework available to the ITS community.[3] As the main objective of ASTUS is to allow experimentation with varied pedagogical strategies in the context of problem-solving tutors, much work has been focused on its foundations, the domain-independent expert and interface modules. Although we have implemented a basic knowledge-tracing (Corbett and Anderson 1995) algorithm that fits ASTUS's hierarchical knowledge representation and a prototypal pedagogical module as a customizable expert system, we first made sure to support a complete inner loop [**Erreur ! Source du renvoi introuvable.**] that can show ASTUS's potential. The next steps include developing these pedagogical and learner model modules to offer basic services and fully show the benefits of modeling a task domain with ASTUS, as well as developing authoring tools that will make modeling easier.

19.3.1 Knowledge Representation

The knowledge representation system in ASTUS is derived from preliminary work on MIACE (Mayers et al. 2001), a cognitive architecture inspired by ACT-R that

[3] An alpha version, limited to internal usage, has been completed and a beta version designed to be shared is under active development (http://astus.usherbrooke.ca).

proposes original twists useful in the ITS context (Fournier-Viger et al. 2006a; Fournier-Viger et al. 2006b; Najjar et al. 2001). Using ASTUS's knowledge representation system, a task domain's declarative knowledge is divided into semantic (factual) and episodic (autobiographical) components, whereas procedural knowledge is modeled at three different grain sizes. First, *complex procedures* are dynamic plans generating a set of goals (intentions satisfied by procedures), according to an algorithm (e.g., sequence, condition, iteration). For example, in the subtraction tutor, the complex procedure *SubtractWithBorrow* is a partially ordered sequence:

```
SubtractWithBorrow (TopSmallerColumn c) {
 subgoal[1] := BorrowFrom(query{nextColumn(c)})
 subgoal[2] := BorrowInto(c)
 subgoal[3] := GetDifference(c)
 order-constraints {(2, 3)}
}
```

Second, *primitive procedures* represent mastered abilities that correspond to steps in the learning environment. Here is one of the two primitive procedures in the subtraction tutor (the other is *ReplaceTerm*):

```
EnterDifference (Column c, Number dif) {
        c.difference := dif
}
```

Third, *queries* and *rules* represent basic or mastered mental skills, such as pattern-matching and elementary arithmetic. Along with complex procedures, they represent mental inferences. As queries and rules define how procedural knowledge components can access semantic ones, they are described in more detail in the discussion of this below.

A class of problem is associated with a goal (in the subtraction tutor, the *SubtractInColumns* goal) that can be satisfied by different procedures, complex or primitive, some of which may be marked as incorrect to represent pedagogically relevant errors (for instance, in the subtraction tutor, the incorrect procedure *AddInsteadOfSubtract*). As complex procedures specify sub-goals, the resulting graph has a goal as root and a set of primitive procedures as leaves.

Aside from goals, which define a semantic abstraction over procedures, semantic components include *concepts*, *relations*, *functions* and *contexts*, and their corresponding instances, *objects*, *facts*, *mappings* and *environments*. Concepts represent pedagogically relevant abstractions and are defined using both *is-a* relationships and essential features. Functions and relations, respectively, represent single- and multi-valued non-essential associations between objects. For example, the subtraction tutor includes these semantic knowledge components:

```
Concept Column {
  position [integer]
  minuends*[Minuend]
  subtrahends*[Subtrahend]
  difference[integer] (the value is initially unknown)
} *a tuple where the first is the current one.
```

```
Function nextColumn {                      Concept Term {
 column[Column] (argument)                   units[integer](0-9)
 next[Column] (image)                        tens[integer](0-1)
}                                                      }

Concept Minuend isA Term                   Concept Subtrahend isA Term
```

Contexts reify the subdivisions of the learning environment, which generally correspond to windows in the interface. Examples of multi-context learning environments include “wizards” and pop-up dialog boxes (the subtraction tutor has only one context). Thus an environment contains all the instances (objects, facts, mappings) related to a distinct subtask of the task domain contained in a context. Domain-level (vs. task-level) objects that represent constants (e.g., the integers 0-9 in the subtraction tutor) are part of a global context that is automatically handled by the framework.

The episodic knowledge components are instances of the semantic and procedural knowledge components that represent the learner’s solution path. The current episode is a graph that contains procedures in progress, done or undone, next possible primitive procedures and planned goals that have not yet been developed. Goals and procedures are specified with parameters and queries. The values of the former come from the parent component (either a goal or a procedure instance) and the values of the latter come from the current environment, according to domain-independent requests such as “get the unique object representing a given concept” or “get the image of a given function”. For example, in the procedure *SubtractWithBorrow* (detailed above) a query fetches the image of the function *nextColumn* for a column specified as a parameter. Queries can also inspect the current episode. For example, in the subtraction tutor, a procedure inspects it to see whether a *BorrowFrom* goal has been satisfied or not.

Rules are used to make relations and functions operational and to classify objects (adding extra is-a relationships). Implemented using Jess, rules help to abstract many of the domain-specific computations that are not relevant in the tutoring process. For example, in the subtraction tutor, the *nextColumn* function is made operational with this rule:

```
(defrule nextColumn
  (Column ?column (position ?p))
  (Column ?next (position ?next:&(= next (+ ?p 1))
  =>
  (Instantiate Function "nextColumn" ("next", ?next)
                              ("column", ?column)))
```

and a column is classified as a TopSmallerColumn with this rule:

```
(defrule TopSmallerColumn
 (Column ?column)
 (CurrentSubtrahend (column ?col) (subtrahend ?subtrahend))
   (CurrentMinuend (column ?col) (minuend ?minuend))
   (test (< (getValue ?minuend) (getValue ?subtrahend)))
   =>
   (Classify ?col "TopSmallerColumn"))
```

Before each of the learner's steps, the episodic graph is developed following each applicable complex procedure's plan, further specified by its arguments, to find the set of next possible primitive procedure. If a step committed by the learner, is included in this set, the graph is updated accordingly, in the other case, the step is added to the off-path steps stack. The graph is not updated while there is a least one step on the stack. Either the tutor or the learner can undo off-path steps, allowing resuming a problem-solving path that can be traced (i.e., not necessarily a correct one). Thus, the episodic components form an interpreted high-level log of the learner's steps, which are stored in a low-level log to allow an exact replay. The latter is required because even if the arguments collected from the learner's interaction in the learning environment match the arguments of a next possible primitive procedure, they may not be exactly the same. For example, the match can be limited to check for a common concept.

19.4 Methodology

In this study, we use incorrect procedural knowledge of the multi-column subtraction domain to add knowledge components to the model and thus yield more data to enrich our comparison. It is uncertain whether being able to give a detailed diagnosis of errors is useful in tutoring (Sleeman et al. 1989), but if it is not supported, it is not possible to conduct studies to evaluate the gains or lack thereof.
The methodology we followed comprises six steps. This section describes each step and its relevance to our comparison of the ASTUS and CTAT frameworks.

19.4.1 Choice of the Task Domain

We chose to model multi-column subtraction because it is representative of well-defined tasks in well-defined domains. Indeed, the arithmetic procedure of subtraction is well documented in the literature and is a clear and precise algorithm. Even though it is well-defined, the subtraction domain contains many inferences that must be deduced by the tutor from steps that may occur in a non-strict order. These characteristics give enough complexity to the solving algorithm to produce an interesting model that can be used to compare the features of the CTAT and ASTUS knowledge representation systems.

VanLehn (1990) assembled a list of 121 procedural errors that can occur when a learner subtracts numbers. Incorporating these errors in the tutors adds knowledge components to implement in CTAT and ASTUS, thus introducing new and possibly complex situations for our models. These situations give us insights about the strengths and weaknesses of the two knowledge representation systems.

19.4.2 Framework-Independent Procedural Model

Once we had chosen multi-column subtraction as the task domain for our comparison, we created a procedural model independent of any tutoring framework.

This model is based on a procedural model presented by Resnick (1982) and was used as a reference when we implemented the CTAT and ASTUS tutors. We chose this model as our reference because it can resolve any multi-column subtraction problem, and it is explicit regarding the inferences and steps that must be made and taken by the learner. To make this model more suitable for a tutoring context, we modified it slightly by adding a new way to subtract a column and including more detail in the borrowing section of the algorithm. The borrowing part of the algorithm was not deemed to be a natural extension of the semantic knowledge we assume the learner know (the base-ten numeral system's basis).

More specifically, the first modification made to Resnick's model is a new way to subtract the current column when it does not contain a number in its subtrahend (i.e., equivalent to a zero). In this case, instead of doing a subtraction, the algorithm simply copies the minuend in the difference section below the line. This modification was taken from a set of subtraction rules given by Brown and VanLehn (1982) and is important in our model, since subtraction errors sometimes depend on the presence or absence of a subtrahend.

The second modification is applied to the original model's borrowing algorithm, to more closely capture the semantic knowledge of the domain in a procedure. When borrowing across zeros, our model does not change zeros to nines from right to left. Instead, it finds the first term which is not zero, decrements it and then iterates from left to right, changing zeros to nines. This procedure more closely represents the semantics of the base-ten numeral system: when borrowing one unit of the next column (to the left) is subtracted to add ten units to the current column. In the case of borrowing across a zero, ten units are borrowed into and one is borrowed from the column, thus changing the zero to a nine.

19.4.3 Error Modeling

When solving problems using arithmetic procedures, systematic errors can be explained by the application of a faulty method (VanLehn 1990). Hence, for each of the 121 errors, we defined the modifications that are needed to our framework-independent model in order for it to be able to produce this error. Each error was modeled the same way: based on the error's definition, we established the modifications that had to be made to the correct model in order to recreate it. Once a model had been generated for all of the errors, we had to apply modifications to the implementation of both tutors. Also, since all the errors were successfully incorporated in our framework-independent model, we can conclude that a failure to implement an error in the CTAT or ASTUS framework is due to limitations of the framework and not because the error cannot be modeled in our problem-solving procedure.

19.4.4 Subtraction Interface

Like the procedural model, the graphical user interface of the tutors was designed independently of the tutoring framework. It was inspired by the interface of

POSIT (Orey 1990), a subtraction ITS that diagnoses learners' errors while they are solving problems (Orey and Burton 1990) and the traditional pen and paper approach. Our interface imposes as few limitations as possible on how learners interact with the learning environment, in order to allow them to express each of the 121 errors.

The interface we designed (shown in Figure 19.3) allows two types of steps: 1) entering the difference for a column; and 2) replacing terms (minuends or subtrahends) in order to borrow from or borrow into a column. The UI actions used to trigger these steps are similar: the learner clicks on a column's difference input box or on the term he or she wishes to replace, and then enters a value using the numerical pad on the right. Once the "OK" button is pressed on the numerical pad, the problem display on the left is updated accordingly.

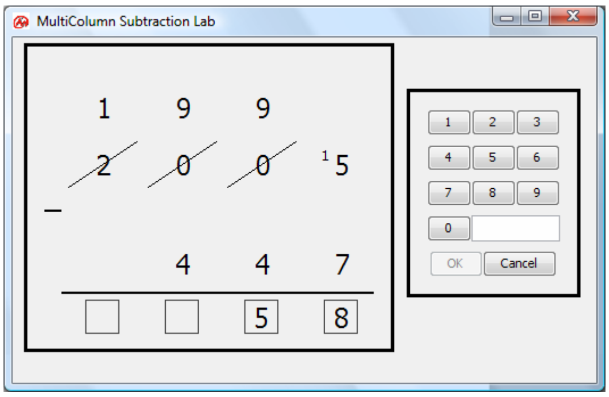


Fig. 19.1 The graphical user interface for the subtraction tutor

19.4.5 Implementation of the Tutors

To implement the CTAT and ASTUS tutors, we began by covering only the knowledge required for a completely correct problem-solving path. As with the framework-independent procedural model, errors should be incorporated without modifying the model already implemented in the tutors.

The model we took from Resnick (1982) accurately describes one way of executing the subtraction procedure, but we had to loosen some of its restrictions to make it applicable in the ITS context, where the model is primarily used to trace the learner's solving path. In particular, the steps that must be executed to subtract two numbers in column can be applied in many different orders while still obtaining correct results, so it is important to consider the multiple ways in which the learner can solve a problem. To be able to correctly trace the learner's solving path, we loosened the borrowing process to remove the constraint on the order of the required steps. For instance, the steps of changing a zero to a nine, decrementing a column and adding ten to a column can be executed in any order. Also, once the current column has been incremented by ten, it can be subtracted even if the borrowing process is not yet finished. Even though we gave learners more freedom, we kept the restrictions on the currently subtracted column so that they must complete all the steps relative to its subtraction before moving to the next one. This restriction has no impact on error modeling and although it is possible to subtract any column as long as subsequent borrows do not affect it, the ordered column sequence from right to left is a key part of the tutored subtraction algorithm.

The model created for our CTAT tutor is organized so that its production rules implicitly replicate a "while loop" in which each of the columns of the problem is subtracted individually, starting from the rightmost and ending with the leftmost. To trace the subtraction of individual columns, our model implements a sub-goal system. The use of such a system is common in examples of tutors given with the CTAT framework and it has the advantage of being flexible with regard to the order of the learner's steps. Sub-goals are normal WMEs that are added to the working memory with the purpose of indicating the steps that are currently possible. In our CTAT model, sub-goals are added to the working memory in the order defined by the framework-independent procedural model. An executed step can thus be accepted if it matches one of the current sub-goals, even if it does not follow the usual ordered sequence of the algorithm. The sub-goal associated with this step is then removed from the working memory to prevent it from being executed again. Once all of the sub-goals for a column have been successfully achieved, the model finds the next column and restarts the process until there is no column left to be subtracted, in which case a sub-goal is added calling for the learner to click on the "Done" button to indicate that he or she considers the problem solved. The "Done" button can also be clicked on when all the columns left to be subtracted have a difference of zero, since these extra zeros are not significant.

ASTUS's model consists of an ordered "for-each" iteration procedure on the columns of the problem. Thus, as in the CTAT model, each column is subtracted individually, from right to left. As each column is subtracted, the episodic graph is developed to obtain the set of primitive procedures that can be executed; the restriction on the order of execution of these primitive procedures is already contained in the complex procedures used to model the column's subtraction. To allow the learner to solve the problem with reasonable freedom, we ensured that the complex procedures contained only those order constraints that are required (borrowing into a column before subtracting it) or pedagogically relevant (subtracting only one column at a time). The problem is considered solved when the "for-each" iteration ends (when the last column of the iteration has been subtracted), or when the "Done" button is pressed (the idea was borrowed from CTAT, but the actual implementation is different because it is handled automatically by the framework), to let the learner finish the problem as soon as all the remaining columns have a difference of zero.

19.4.6 Incorporation of Errors

Once we had completed the basic implementation of the two tutors, we started improving them by incorporating the errors we had modeled. Of the 121 documented errors (VanLehn 1990), we implemented 46. The first 26 errors were modeled by systematically incorporating each error, in the same order we followed in our framework-independent model. After those errors were completed, we had enough experience to evaluate the challenges incorporation of specific errors would pose. Of the 95 remaining errors, we implemented 20 that were more challenging and required modeling behaviors that had not been previously encountered. The remaining 75 errors were not incorporated in the tutors because they were similar to the ones previously implemented and thus did not pose new challenges.

Both frameworks allowed the successful incorporation of the 46 selected errors, but the effort required showed the strengths and weaknesses of the CTAT and ASTUS knowledge representation systems. The "Results" section explains these strengths and weaknesses and the features from which they arise.

19.5 Results

As both frameworks allowed us to produce a comprehensive model of the subtraction domain, we need to show: how each framework's features have influenced the modeling process of the tutors and which type of pedagogical interactions are offered by each of the frameworks.

19.5.1 Modeling Process

The results concerning the modeling process are presented in five sections. The first gives details on how ambiguous steps are managed by the two frameworks. Then we compare how specific error types are modeled and handled; in particular, we discuss errors containing multiple steps and errors containing correct steps. Next we present how each framework allows the reuse of knowledge components. Finally, we examine the coupling between the interface and the model and how this influences the implementation of the tutor.

19.5.1.1 Ambiguities

When tracing a learner's solving path, situations may arise in which different procedures yield the same steps. These steps are qualified as being ambiguous because they can be interpreted in more than one way. Ambiguities may be present in an error-free problem-solving path, but in the subtraction tutors, they result from the inclusion of errors. When we added errors to our tutors, we found three kinds of situations that led to ambiguities. First, two unrelated errors can lead to the same step for specific problems. For instance, the *add-instead-of-sub* (adding the term of a column instead of subtracting) and *smaller-from-larger* (subtracting the smaller number from the larger regardless of which one is the minuend) errors both result in the learner entering 2 as the difference when they are applied to a "5 – 7" column. Second, errors can involve the same behavior but in different application conditions, thus resulting in ambiguities when many sets of conditions are satisfied at the same time. For an example of this situation, consider the errors *diff-0 – N = 0* (writing 0 as the difference when subtracting a number N from 0) and *0 – N = 0-after-borrow* (the same behavior but occurring only when the column has already been borrowed from). Third, ambiguities occur when some or even all of the steps of an incorrect procedure can be considered correct. In this case, the tutor must be able to determine which procedure was executed when an unambiguous step is taken, or give priority to the correct procedure when it is not possible to decide whether the error was committed or not. An example of this situation is the *borrow-skip-equal* error (skipping columns where the minuend and subtrahend are equal during the borrowing process), in which correct steps are omitted rather than incorrect ones being performed. Details of this error are presented in the "Errors containing correct steps" subsection.

Using the model tracing algorithm of CTAT, ambiguities are resolved according to rule priority. When a step is executed, a search is performed to determine which rule path can explain it. Since the execution order of this search is based on the priority of each rule, the rules of the matching path with the highest priorities will be discovered first and will be fired, preventing the discovery of any ambiguities that might have occurred. Additionally, buggy rules always have lower priorities than valid ones, so a step that can be evaluated as correct will be so evaluated even if an incorrect path exists.

ASTUS's hierarchical procedural model allows a tracing algorithm in which an episodic graph containing all the applicable procedures is updated before each of the learner's steps. This feature allows ASTUS to detect and handle ambiguous steps. The graph is used to select the appropriate explanation for previous steps where evidence resolving the ambiguities was found. Evidence includes the execution of a non-ambiguous step or the signal sent by the "Done" button. In the former case, the framework uses the episodic graph to find the nearest common ancestor of the procedure actually executed and the ambiguous one. In the latter case, the graph is searched for a procedure that is completed and that can satisfy the root goal.

In summary, CTAT prevents ambiguous states by always firing the production rules with the highest priority. This method is easy to implement and handles most of the ambiguity in a way that has no negative effects on the tracing process. It is possible to introduce special treatment of ambiguities by adding extra production rules to the model. An example is given in the "Errors containing correct steps" subsection below. CTAT is thus capable of handling any ambiguity, but the drawback is a more complex model containing superfluous knowledge components. With ASTUS, all ambiguities are handled directly by the framework's tracing algorithm, thanks to its top-down hierarchical procedural model and the resulting episodic graph. This approach requires more effort when developing the framework, but authors do not have to worry about ambiguities. Both ASTUS and CTAT must deal with permanent ambiguities that no evidence can resolve, either by using priorities or by relying on the learner model.

19.5.1.2 Errors with Multiple Steps

Some errors require multiple steps from the learner for complete diagnosis. This is the case for errors where the whole problem is solved incorrectly, such as *add-no-carry-instead-of-sub* (the learner executes an addition without carrying instead of

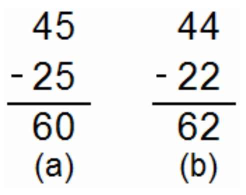


Fig. 19.2 Examples for the add-no-carry-instead-of-sub error. (a) The first column is ambiguous. (b) The error should not be diagnosed

a subtraction). As well as involving multiple steps, these errors can also produce ambiguities. For instance, in the *add-no-carry-instead-of-sub* error described previously, columns where the sum of the minuend and the subtrahend is the same as their difference can exist. An example of this would be the 15 – 15 problem where the sum and the difference of the unit column are both 0 and, because the carry is not performed, there is no way of knowing whether the learner intended to add or subtract the columns (i.e., the ambiguity is permanent).

When it comes to reacting to the learner's errors, CTAT implements a policy of immediate feedback in which there is no delay between the execution of the error and the tutor's feedback. Thus, CTAT's error detection only allows one step to be executed before feedback is given; this poses particular challenges when dealing with errors containing more than one step. For instance, when dealing with errors that are composed of multiple steps that can be executed in different orders, since there is no way of knowing which one the learner will execute first, all of them need to trigger the error diagnosis. Also, because the learner does not have the opportunity to complete all of the steps in the error, CTAT has less evidence that the error it diagnosed is really the one being made. Other challenges come from errors that require an ordered sequence of steps. In those cases, since only one step can be executed before feedback is given, the tutor should diagnose the error when the first step of the sequence is executed. This is difficult when the first steps are ambiguous. Figure 19.2 illustrates such a situation in the case of the *add-no-carry-instead-of-sub* error. If the error diagnosis is made on the basis of the first column alone, the error is not detected in Figure19.2 (a) since the first step is ambiguous. On the other hand, if we allow the error to be diagnosed on any column, it will be detected in Figure 19.2 (b) even if the first column was correctly subtracted[4]. To achieve accurate diagnosis in every situation, this ambiguity must be handled in the procedural knowledge, in a way that is specific to this error. Thus, when an addition is detected, the model needs to iterate on all the columns that were previously processed to see whether the entered results can be interpreted as either the subtraction or the addition of each column. If an ambiguity is found for each of the preceding columns, then the *add-no-carry-instead-of-sub* diagnosis can accurately be given.

[4] Since the learner correctly subtracted the first column we can assume that he/she can subtract (without borrowing) and that the second error is not due to the *add-no-carry-instead-of-sub* error.

Even if we overlook CTAT's immediate feedback policy, it would still be challenging to handle errors with multiple steps because production rules are independent of each other. The lack of an explicit hierarchy in the procedural model has two important consequences: it is 1) difficult to identify that multiple steps are caused by the same occurrence of an error and 2) difficult to evaluate how many steps are caused by an occurrence of an error. These two characteristics prevent the system from delaying feedback until the learner completes all of the steps in an error.

In ASTUS, an error with multiple steps can easily be modeled by creating an incorrect complex procedure. The model's hierarchy allows the creation of an episodic graph in which the primitive procedures caused by a specific occurrence of an error are easily identified. It is thus possible for the tutor to associate multiple steps with a specific error occurrence.

19.5.1.3 Errors Containing Correct Steps

There are situations where an error is not described by the incorrect steps it causes but rather by the correct ones that are skipped when it is made. Being able to handle these errors is a specific case of ambiguity management, and thus the features of each framework that make it possible or challenging are the same ones described previously in the ambiguity subsection. An example of such a situation is the *borrow-skip-equal* error (skipping columns where the minuend and subtrahend are equal during the borrowing process). An example of its execution is shown in Figure 19.3, where all of the executed steps for the first column are correct and the error can only be diagnosed when the learner subtracts the second column and enters zero as its difference. Furthermore, in the situation shown in Figure 19.3 (a), there are four correct steps that must be performed before the next column can be used to diagnose the error. It is essential that the exact sequence of steps is executed before diagnosing the *borrow-skip-equal* error since, as shown in figures 19.3 (b) and 19.3 (c), the exclusion or addition of one could change the diagnosis. In 19.3 (b), five correct steps have been performed and one of them was changing a zero to a nine in a column where the minuend and the subtrahend are equal. Because of this step, we know that the error made is not *borrow-skip-equal*, but probably a slip where the learner forgot one of the columns. On the other hand, Figure 19.3 (c) shows a situation where only three of the four required steps have been executed. This shows why it is mandatory to check for the presence of the exact sequence of steps defined by the error; in this case, removing one of them can completely change the diagnosis. The solving path shown in this figure could be associated with either the *borrow-across-zero* (not borrowing on zeros and skipping to the next column) or the *always-borrow-left* (always borrowing from the leftmost column) errors.

In CTAT, we need to be able to determine when entering the difference of the next column will cause the error to be diagnosed. Since all of the correct steps are already covered by existing production rules, we have to add rules in order to 1) mark all steps that would be performed if the learner was executing the error and 2) iterate through the columns to check that each of these steps was executed. Handling errors containing correct steps is similar to handling errors with multiple

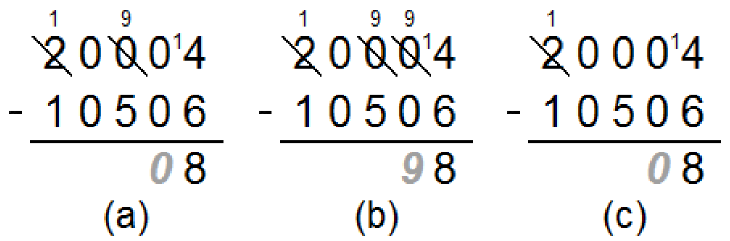


Fig. 19.3 An example of the borrow-skip-equal-error. (a) The error can only be diagnosed when the difference is entered in the second column. (b) When an additional step is present, we cannot evaluate the error as borrow-skip-equal. (c) One of the steps is ng and the error could either be borrow-across-zero or always-borrow-left

steps: the mechanism used to manage the ambiguities must be implemented in the model. Implementing such a mechanism can be complex and tedious: for the *borrow-skip-equal* error, 11 production rules are required to give a correct diagnosis, while only 14 are required to trace an error-free subtraction solving path. Adding ambiguity management in the model for this error requires almost as much effort as implementing the complete model for the basic subtraction tutor. Thus, one advantage of a framework with a knowledge representation system similar to ASTUS's is its built-in approach to manage ambiguities, significantly reducing the effort needed to model complex errors.

Modeling this kind of error in ASTUS does not cause any difficulty: an incorrect complex procedure containing an ordered sequence of sub-goals is created, with a last one implicitly indicating that a part of the correct procedure was skipped. In the case of the *borrow-skip-equal* error, the last sub-goal is to subtract the next column. The main difference is that in ASTUS the error is part of the domain, whereas in CTAT it is recognized using additional rules but not explicitly modeled (e.g., the tutor cannot generate the results of this error to demonstrate it).

19.5.1.4 Reuse of Knowledge Units

Both CTAT and ASTUS allow the reuse of knowledge components in different situations. In this section we give examples of how each of the frameworks reuses knowledge components in the subtraction domain. Both knowledge representation systems allow an author to reuse previously defined procedural and semantic knowledge components to model errors. Reusing existing components reduces the complexity of the resulting model and decreases the effort needed to implement it. An example of how the modeling process can be simplified is taken from the *always-borrow* error, in which the learner systematically borrows before subtracting a column even if it is not necessary. To implement this error, we simply reuse the borrowing procedure that has been previously modeled, by forcing its use on a column that would not require it.

In CTAT, procedural knowledge can be reused with the help of buggy production rules that do not produce a step. These rules can be used to alter the content of the working memory in order to create a rule path containing valid rules ultimately flagged as incorrect. In our model, we use previously defined sub-goals to reproduce existing behaviors in an erroneous context. For instance, with the *always-borrow* error, a buggy rule adds existing sub-goals relative to the borrowing process (i.e., decrementing the next column, adding ten to the current column) in a situation where they are not required. The equivalent can be achieved in ASTUS by defining incorrect complex procedures that use existing goals. Hence, procedures that had been previously defined are used to describe erroneous behaviors. For example, the *always-borrow* error is modeled by an incorrect procedure using the goals of borrowing from and borrowing into a column.

Error composition happens when multiple erroneous behaviors are present at the same time in a solving path. In both CTAT and ASTUS, the reuse of correct procedural knowledge in modeling pedagogically relevant errors allows the recognition of composite errors because the correct procedures that are reused can themselves have erroneous alternatives. The "Pedagogical interactions" section in 5.2 shows an example of a learner displaying both the *always-borrow* (borrowing even if not required) and the *borrow-from-bottom* (borrowing from the subtrahend) behaviors.

In CTAT, production rules test conditions on existing WMEs and perform computations to modify or create new ones. In ASTUS, the complex procedures follow a fixed, domain-independent behavior so that domain-specific conditions and computations are not directly included in them, but added indirectly via queries and rules. For example, the mapping between a column and the number of zeros to its left can be used by both the *borrow-decrementing-to-by-extras* (when borrowing into, increment by 10 minus the number of zeros borrowed across) and *decrement-by-one-plus-zeros* (when borrowing from, decrement by 1 plus the number of zeros borrowed across) errors. In CTAT, this behavior can be emulated with highly prioritized production rules that perform computations and store the result in WMEs. In many cases, including the errors cited above, high priority of the rules is necessary because the computations must be performed before the steps of the borrowing process have been executed, even though the buggy rule can be executed after them. For instance, in the *decrement-by-one-plus-zeros* error, decrementing the minuend of the column that is borrowed from can be executed after the zeros have been changed to nine. It is then crucial that the number of zeros to the left of the current column be counted before the terms are changed. Another example of the reuse of knowledge components in ASTUS is the classification of a column object as having "been borrowed from". Some errors occur only for these columns and the procedures modeling them require the column to be an instance of the "column borrowed from" concept. In CTAT, WMEs can contain slots with a boolean value to emulate classification. In our subtraction model, this is indeed the case. Such a slot must be manually updated in every production rule that could change its value, whereas in ASTUS, classification is automatically retested when an object is modified.

19.5.1.5 The Coupling between the Model and the Interface

In this section we detail how each framework manages its user interface, based on examples taken from our subtraction tutors. The two frameworks have opposite approaches in terms of how they link the user interface and the model. In CTAT, the interface and the model are almost entirely separate: the interface has no access to the content of working memory and the only information concerning the interface that the model receives is through Selection Action Input (SAI) events. A SAI event is sent when a step is performed on the interface, and it contains the element of the interface that triggered the event (selection), the action that has been performed on this element (action) and the value that was entered (input).

Since there is no direct link between the interface and the model, information concerning the problem's current state that is useful for tracing must be stored in WMEs that are updated using the SAI events. For instance, in our subtraction tutor, the working memory contains WMEs representing each of the interface's text fields, to store their value. We must ensure that the values contained in working memory are synchronized with the content shown on the interface by using the data contained in the SAI events.

Another effect of having a weak link between the interface and the model is that the interface's adaptability to the problem data is limited. Since the interface has no access to the content of working memory, it cannot use the problem data to generate its interface elements. Thus, it is difficult to dynamically add interface elements in order to match the problem state. In our subtraction tutor, this means that the interface has a fixed number of columns and a problem cannot contain more or less than that number, as this would require creating (or hiding) the text field dynamically.[5] The severity of this limitation varies according to the particular design of a tutor's interface: for example, it would have been possible to dynamically add columns if we used a "table component"[6] instead of individual text fields for each term.

The ASTUS framework enforces a stronger link between the model of the task domain and the interface (Fortin et al. 2008). Unlike CTAT, ASTUS gives the interface elements access to the instances of the current environment. This is achieved by the use of *views,* scripts that describe the representation of concepts in the interface. This MVC-based approach allows the interface to reflect the content of the current environment at all times. When the effects of a primitive procedure are applied to the environment, the views of the modified objects are notified so that they can update their UI representation accordingly. For example, in our subtraction tutor, when borrowing from or decrementing a minuend, the primitive procedure adds a new minuend to a column object, which view is then notified so that it displays the new minuend and strikes the old one. This approach enables a tutor built with ASTUS to have a flexible interface that adapts itself dynamically. For this reason, the ASTUS subtraction tutor can have the right number of columns to fit any problem.

Each primitive procedure is associated with an *interaction template* that triggers the execution of a step when the required basic UI actions are matched (Fortin et al. 2008). These templates offer a solution to the difficulty which CTAT tutors circumvent by "tutorable" interface elements (for example, a panel

[5] It is possible for a production rule to call static Java methods to produce side-effects on the tutor's interface; a sample "Truth tables" tutor uses this technique.

[6] This is the solution used in sample addition and subtraction tutors.

containing multiple combo-boxes and a button). Instead, with our approach, steps can be triggered by multiple interactions on multiple interface elements. The most important benefit of this approach is its capacity to generate the interactions by using the template and the episodic graph to produce a step with complete visual feedback (i.e., mouse moves and clicks). It is then possible to use the demonstration of a step as pedagogical feedback. There is an obvious cost for supporting this form of feedback, but we found out that it also helped us come up with more comprehensive models, as data implicitly defined by the interface must be explicitly encoded to define the templates. In some specific cases, we may implement multiple interactions in a single component if decomposing the step is not pedagogically relevant. For example, the "numerical pad" of our subtraction tutors is a single component.

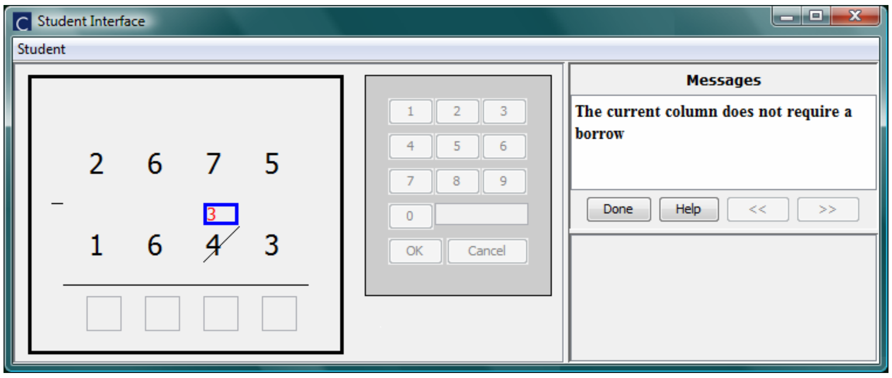
We believe an MVC approach to the model-interface coupling has many advantages over a weaker one. It allows us to adapt the interface to the content of a particular problem, it prevents synchronization issues between the interface and the environment and it allows sophisticated pedagogical interactions such as demonstration. On the other hand, a weaker link such as the one offered by CTAT is easier to implement and, more importantly, makes it easier to develop tools that can be used to create the tutors' interfaces. These authoring tools are important, since they can greatly reduce the effort required to create a new tutor, but they are always more effective when dealing with rather simple or formatted interfaces.

19.5.2 Pedagogical Interactions

In this section we show how each of the frameworks gives pedagogical feedback. We start by describing the different types of interactions that are supported by at least one framework. We then use four scenarios taken from the subtraction tutors to give examples of how CTAT and ASTUS behave when reacting to learners' steps. Table 19.1 (presented at the end of this section) summarizes how the individual interaction types are supported by each framework.

19.5.2.1 Pedagogical Interaction Types

- Immediate feedback: minimal feedback to indicate whether a step is correct or incorrect. This includes flag feedback: changing the color of an input value to indicate whether it is correct or incorrect.
- Interface highlights: focusing the learner's attention on a specific part of the interface, for example by painting a rectangle over it.
- Next-step hint: giving a hint towards one of the next correct steps.
- Error-specific feedback: giving feedback regarding an error that is contained in the model.
- Off-path error recognition: recognizing certain off-path steps and giving feedback accordingly.
- Demonstration: demonstrating, with full visual feedback, how to perform a step (i.e., the tutor takes control of the mouse and keyboard).



The current column does not require a borrow

Fig. 19.4 An example of CTAT's reaction to a composite error

19.5.2.2 Scenario 1

The first scenario illustrated in Figure 19.4 shows a case of error composition: solving the problem 2675 – 1643, the learner changes the 4 (subtrahend of the second column from the right) to a 3. This action is caused by the composition of two errors: *always-borrow* (borrowing even if it is not required) and *borrow-from-bottom* (borrowing from the subtrahend). For both frameworks, the recognition of composite errors is made possible by reusing procedural knowledge, but the feedback they produce is different.

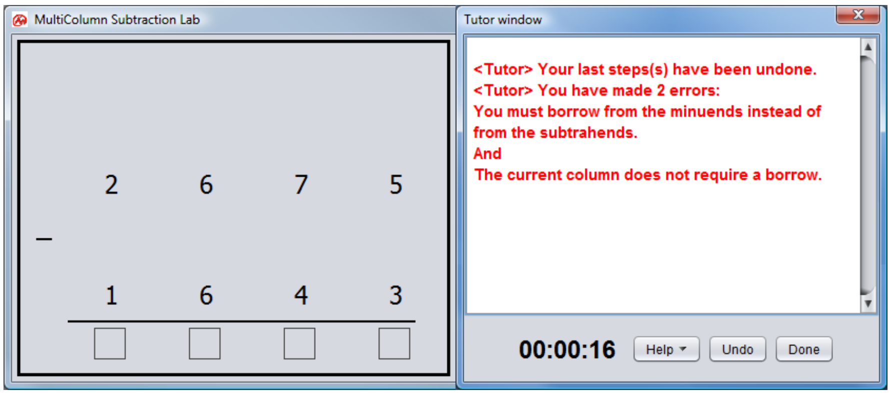
In this situation, the CTAT tutor reacts (see Figure 19.4) by: 1) flagging the incorrect value entered by writing it in red; 2) highlighting the replaced term by drawing a blue frame around it; and 3) giving a message related to the error. This message is produced from a template associated with the first buggy rule encountered (*always-borrow*) while the second (*borrow-from-bottom*) is ignored.

In the same situation, the ASTUS tutor reacts by first undoing the learner's step and updating the interface accordingly; and second, displaying a message that indicates which errors have been traced. As shown in Figure 19.5, in cases of error composition, ASTUS's written feedback includes all of the errors found.

19.5.2.3 Scenario 2

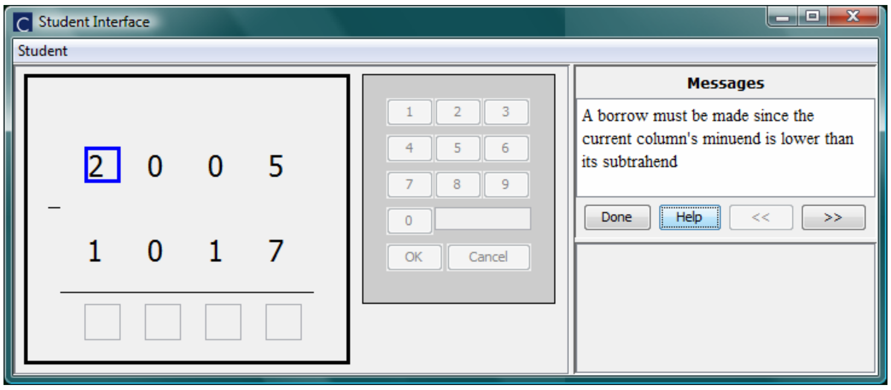
The second scenario shows how the tutors interact with the learner when a next-step hint is requested. We show how the learner's solving path affects the chosen hints by illustrating how the tutors react to two hint requests. These requests lead to the same step but at two different points in the solving path of the 2005 – 1017 problem. The first hint request is executed when no step have been yet performed by the learner (Figure 19.6) while the second one is made when the only remaining step is to decrement the minuend (2) from the leftmost column (Figure 19.7).

The method used by CTAT in providing hints is to 1) find the rule chain that produces the next step; 2) generate messages using the templates associated with



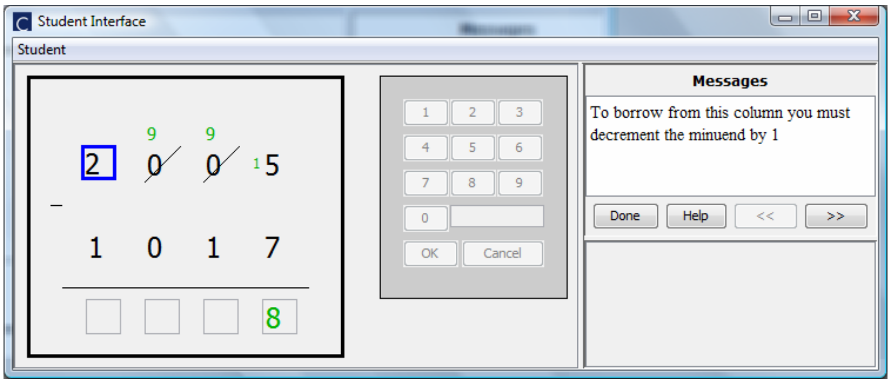
<Tutor> Your last steps(s) have been undone.
<Tutor> You have made 2 errors:
You must borrow from the minuends instead of from the subtrahends.
And
The current column does not require a borrow.

Fig. 19.5 An example of ASTUS's reaction to a composite error



A borrow must be made since the current column's minuend is lower than its subtrahend

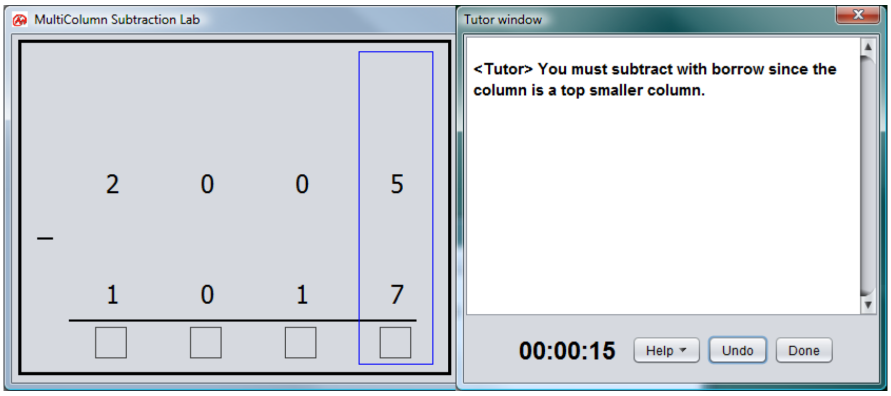
Fig. 19.6 CTAT's feedback when a hint is requested at the beginning of the problem



To borrow from this column you must decrement the minuend by 1

Fig. 19.7 CTAT's feedback when a hint is requested and only the decrementing remains

each of the rules and display them in the same order the rules were fired in (the arrow buttons can be used to navigate the hint list); and 3) highlight the interface component on which the action must be executed. In this scenario, the two hint requests generate different production rule chains and thus the first hint message displayed differs depending on the solving path. For the first request, the hint given concerns the condition required to borrow (Figure 19.6) and, for the second, the hint concerns decrementation of the minuend (Figure 19.7). In both cases, even if the messages displayed are different and refer to different parts of the problem, the highlight is applied to the minuend of the leftmost column. In the case illustrated by Figure 19.6, the highlight should focus the user's attention on the current (rightmost) column but, since the highlight is determined by the SAI event contained in the production rules, the displayed highlight does not correspond to the hint message. Even though highlighting is only supported for rules containing SAI events, we see nothing in the knowledge representation system that would prevents its implementation in rules that modify the working memory.



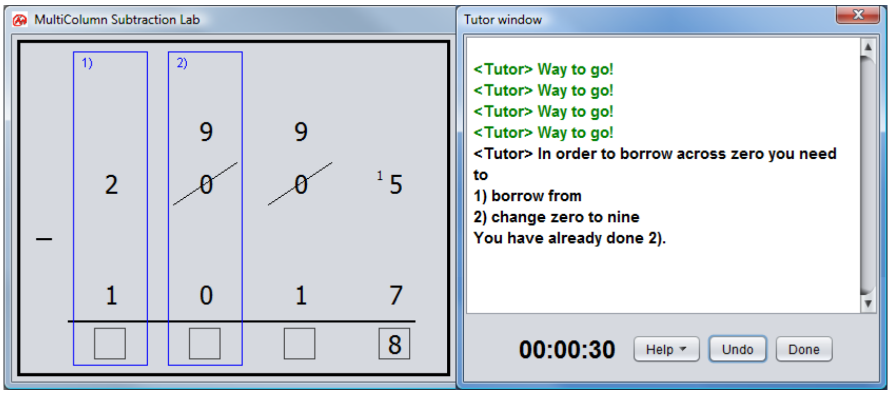
<Tutor> You must subtract with borrow since the column is a top smaller column.

Fig. 19.8 ASTUS's feedback when a hint is requested at the beginning of the problem

In ASTUS, the tutor uses the examinable knowledge components of the model to automatically generate next-step hints.[7] In contrast with CTAT, no message templates are required, although such templates are also supported for situations where customized messages are helpful. Using the hierarchical procedural model, the episodic graph and the learner model, the tutor can evaluate which procedures the learner is most likely to be executing and choose the one on which help should be given. For example, in figure 19.8, the next step is to decrement the minuend of the leftmost column. Since the borrowing process has not been initiated yet, the tutor evaluates that the learner needs help on the conditional procedure which determines whether borrowing is required. A hint is then generated using this procedure's definition. In figure 19.9, the next step is also to decrement the minuend,

[7] Hint generation requires that knowledge components receive meaningful names in all supported languages; internationalization issues may arise.

but the tutor recognizes that the borrowing process has been started and now gives feedback for the sequence procedure used when borrowing across a zero. With ASTUS, hints can be given on partially completed procedures (borrow across zero) while in CTAT, once a rule has been fired, it will never produce a hint again. ASTUS also behaves differently from CTAT in highlighting the interface. The use of views allows the tutor to highlight any object, even if its view is composed of multiple interface components, as is the case for columns. Additionally, ASTUS uses the procedure parameters to determine which components to highlight; highlights are thus supported for every procedure, not only the primitive ones. All objects showing up as arguments that have a view in the current environment may be highlighted, but complex procedures may give special purposes to specific parameters and may use this information to select which ones will be highlighted (e.g., a for-each procedure has a parameter that designates the set to iterate on).



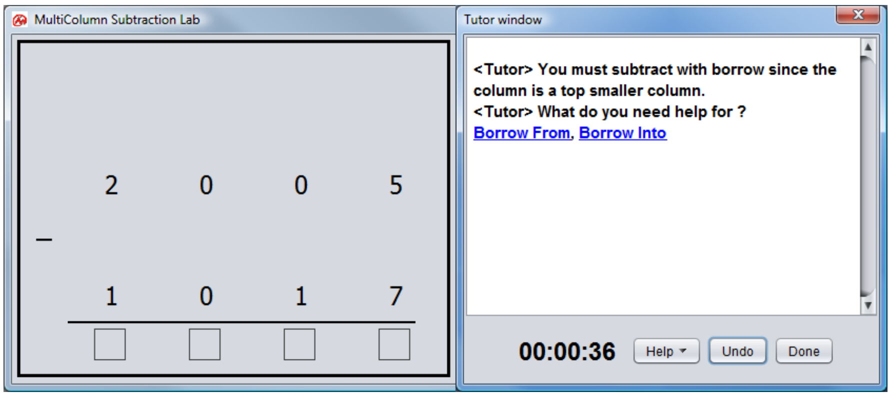
<Tutor> Way to go!
<Tutor> Way to go!
<Tutor> Way to go!
<Tutor> Way to go!
<Tutor> In order to borrow across zero you need to
1) borrow from
2) change zero to nine
You have already done 2).

Fig. 19.9 ASTUS's feedback when a hint is requested and only the decrementing remains

19.5.2.4 Scenario 3

As shown in the second scenario, both CTAT and ASTUS produce multiple messages when a hint is requested. These messages are linked to the different procedural components used to produce the next step from the chain of matched rules in CTAT or the episodic graph in ASTUS. The tutor can provide hints at multiple levels, from more abstract (such as subtracting a column) to more specific (such as entering the difference).

Besides being able to vary the hint level between top-down and bottom-up, ASTUS can also offer help on any of the paths leading to a step. When the learner requests help and multiple different steps can be executed, ASTUS can ask for additional information to determine the best hint to provide. For this purpose, ASTUS uses links similar to the "Explain more" feature employed by Andes (VanLehn 2005). Each link represents a goal the learner can choose to request help on. For example, in the "2005 – 1017" problem, if the learner asks for help



<Tutor> You must subtract with borrow since the column is a top smaller column.
<Tutor> What do you need help for ?
Borrow From, Borrow Into

Fig. 19.10 In ASTUS, the tutor asks the learner which goal he/she wishes to be helped with

twice before performing any step, the tutor asks which step he/she intends to do. In this situation, the tutor would propose the “Borrow From” and the “Borrow Into” goals (Fig. 19.10).

We did not find an equivalent mechanism in CTAT’s Jess-based cognitive tutors, although example-tracing tutors can give hints for different next steps depending on which interface component currently has the focus. A similar behavior could be implemented, as it would consist of searching for a production rule which results in an action that uses the interface component currently being focused on.

19.5.2.5 Scenario 4

The last scenario we present illustrates feedback automatically generated when off-path steps are recognized by the tutor. In this example, the learner tries to solve the “2675 – 1640” problem by starting with the leftmost column and entering 1 as the difference.

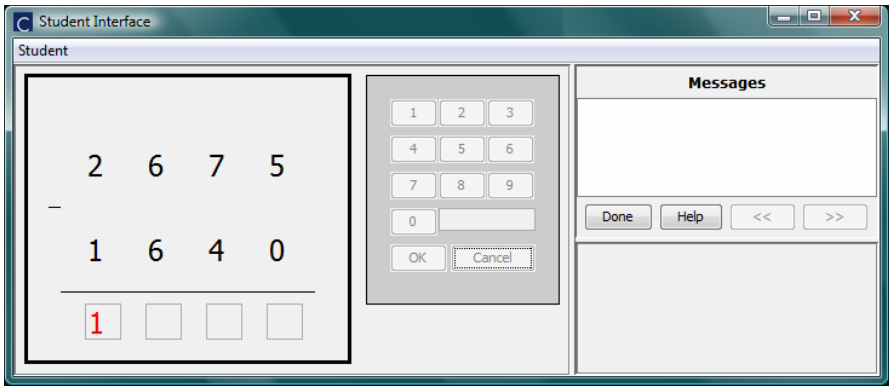
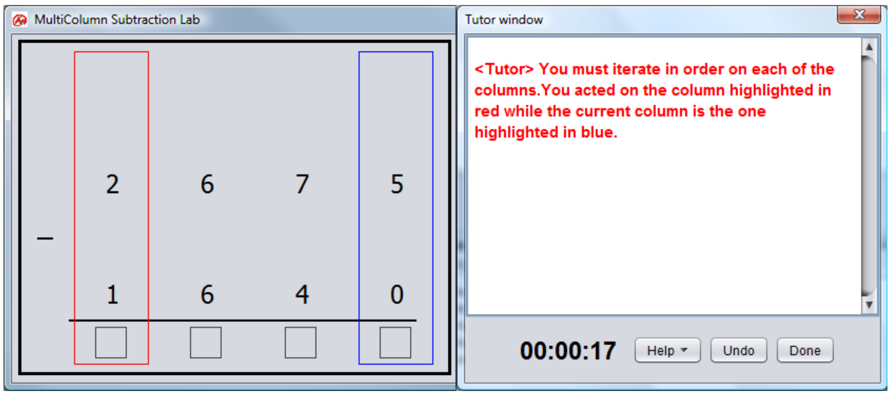


Fig. 19.11 CTAT’s feedback for an step executed in a column other than the current one.



<Tutor> You must iterate in order on each of the columns.You acted on the column highlighted in red while the current column is the one highlighted in blue.

Fig. 19.12 ASTUS's feedback to a step executed in a column other than the current one

Table 19.1 Summary of the supported feedback types

Type of feedback	CTAT	ASTUS
Immediate feedback	Flag feedback Red is used for errors, green for correct steps	Messages in the tutor window ("Way to go!" and "step has been undone")
Interface highlights	Can highlight interface components such as text fields	Can highlight any semantic knowledge viewable on the interface
Next-step hints	Abstract to specific	Abstract to specific Hints on multiple next-steps Next step hint generation
Error-specific feedback	Supported for modeled errors.	Supported for modeled and composite errors.
Off-path step recognition	Not supported	Supported for planning errors such as forgetting an iteration in a "for-each" procedure.
Demonstration	Not supported	Supported

The feedback given by CTAT in this situation is shown in figure 19.11. Since there are no buggy rules that lead to the executed step, CTAT treats it as a non-specific error and only uses flag feedback to indicate that the step is incorrect.

With ASTUS's features, it is possible to implement recognition of errors that go against the scripted behavior of a complex procedure when an off-path step is executed, and to generate error messages accordingly. Figure 19.12 illustrates ASTUS recognizing an error in the iteration on the problem's columns. To make this diagnosis, ASTUS examines the currently available procedures to identify one describing an ordered iteration on a set of columns. It then checks the arguments of the primitive procedure executed by the learner to link one of them to a column on which the iteration is applied. If the identified column is not the one for which steps are currently available, the tutor generates feedback according to a domain-independent predefined template.

19.6 Conclusion

The goal of the study presented in this chapter was to compare the knowledge representation system of the ASTUS framework with a well known production rule-based system such as the CTAT framework. We especially wanted to see whether the two frameworks allow us to model the well-defined task domain of multi-column subtraction in its entirety, what kind of pedagogical interactions they can offer and which situations are more difficult to model in each of them. To achieve our objectives, we compared the modeling process of two subtraction tutors (one in each of the frameworks) into which we incorporated 46 pedagogically relevant errors.

Both CTAT and ASTUS were flexible enough to correctly model the subtraction task domain while still allowing freedom in the order of the learner's steps and being able to model all the errors found in the literature (VanLehn 1990). Even though all errors can be diagnosed by both frameworks, limitations were encountered in incorporating some of them into the models.

Regarding the pedagogical aspect of the tutors, both frameworks can provide immediate feedback, produce hint or error-specific message from templates associated to a procedural knowledge component and highlight elements in the interface. ASTUS has the advantages of being able to: recognize error composition, generate next-step hints or error-specific feedback on off-path steps, offer more sophisticated highlights, demonstrate how a step must be performed on the interface and giving the learners more control over the kind of help they need.

As we incorporated errors into our tutors, we encountered many situations that posed challenges to the modeling process. One challenge that appeared frequently was the management of ambiguous steps. CTAT's solution is simple to implement and ensures that ambiguities are prevented by always selecting the procedures with the highest priority. The author can still implement more complex management by adding production rules to the model. On the other hand, ASTUS handles complex ambiguity cases in its tracing algorithm. Another difficulty that the frameworks must be able to handle is the reuse of knowledge components. In our study, we encountered many situations where previously defined procedural or semantic knowledge components could be reused in order to keep the model as simple as possible. Both CTAT and ASTUS implement mechanisms that allow the author to easily reuse previously defined components. The nature of the problem can also bring difficulties in authoring a tutor In the case of subtraction task domain, the tutor must adapt the interface in function of the number of columns specified for each different problem. With CTAT, it is more difficult because of the lack of a direct link between the model and the interface.

The two frameworks follow different approaches regarding the authoring process of a tutor. CTAT's tracing algorithm and knowledge components are kept simple, as the primary focus has been on decreasing the effort needed to create tutors (Aleven et al. 2006), in order to reduce the time needed by experienced modelers and make the modeling process accessible to non-programmers (Example-Tracing tutors (Koedinger 2003) goes further in this way) . On the other hand, ASTUS is designed to be used by programmers able to manipulate more complex knowledge components. These components allow us to incorporate a number of domain-independent behaviors such as ambiguity management and the generation of next-step hints or error-specific feedback on off-path steps.

With this study, we have shown how ASTUS's knowledge representation system allows us to model complex well-defined task, and that it can be compared to other existing frameworks such as CTAT. There is still much work that can be done to improve our framework. Our efforts have been largely focused on the knowledge representation and our next steps will be to develop other domain-independent modules such as the learner model, the outer loop's task selection and sophisticated domain-independent pedagogical strategies. It would also be interesting for us to do a similar comparative study with a constraint-based framework such as ASPIRE (Mitrovic et al. 2006).This would allow us to evaluate whether our system has advantages over the constraint-based approach and which elements of this approach can be adapted to improve the system we are developing.

Acknowledgments. We would like to thank Vincent Aleven, Ryan S.J.D. Baker and Jean-Pierre Mbungira for their comments and suggestions on our study and its results.

References

Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. In: Ikeda, M., Ashley, K.D., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 61–70. Springer, Heidelberg (2006a)

Aleven, V., Sewall, J., McLaren, B.M., Koedinger, K.R.: Rapid Authoring of Intelligent Tutors for Real-World and Experimental Use. In: Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies. IEEE Computer Society, Los Alamitos (2006b)

Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. International Journal of Artificial Intelligence in Education 19, 105–154 (2009)

Anderson, J.R., Pelletier, R.: A Development System for Model-Tracing Tutors. In: Proceedings of the International Conference of the Learning Sciences, VA, Association for the Advancement of Computing in Education, Charlottesville (1991)

Anderson, J.R.: Rules of the Mind. Lawrence Erlbaum Associates Inc., Hillsdale (1993)

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

Anderson, J.R., Lebiere, C.: The Atomic Components of Thought. Lawrence Erlbaum, Mahwah (1998)

Brown, J.S., VanLehn, K.: Towards a Generative Theory of "Bugs". In: Addition and Subtraction: A Cognitive Perspective. Lawrence Erlbaum, Hillsdale (1982)

Corbett, A.T., Anderson, J.R.: Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. User Modeling and User-Adapted Interaction 4, 253–278 (1995)

Fortin, M., Lebeau, J.F., Abdessemed, A., Courtemanche, F., Mayers, A.: A Standard Method of Developing User Interfaces for a Generic ITS Framework. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) ITS 2008. LNCS, vol. 5091, pp. 312–322. Springer, Heidelberg (2008)

Fournier-Viger, P., Najjar, M., Mayers, A., Nkambou, R.: A Cognitive and Logic Based Model for Building Glass-Box Learning Objects. Interdisciplinary Journal of Knowledge and Learning Objects 2, 77–94 (2006)

Fournier-Viger, P., Najjar, M., Mayers, A., Nkambou, R.: From Black-box Learning Objects to Glass-Box Learning Objects. In: Ikeda, M., Ashley, K.D., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 258–267. Springer, Heidelberg (2006)

Heffernan, N.T., Koedinger, K.R., Razzaq, L.: Expanding the Model-Tracing Architecture: A 3rd Generation Intelligent Tutor for Algebra Symbolization. International Journal of Artificial Intelligence in Education 18(2), 153–178 (2008)

Koedinger, K.R., Aleven, V., Heffernan, N.: Toward a Rapid Development Environment for Cognitive Tutors. In: Proceedings of AI-ED, Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies, pp. 455–457. IOS Press, Amsterdam (2003)

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

Matsuda, N., Cohen, W.W., Koedinger, K.R.: Applying Programming by Demonstration in an Intelligent Authoring Tool for Cognitive Tutors. In: AAAI Workshop on Human Comprehensible Machine Learning, pp. 1–8 (2005)

Mitrovic, A., Koedinger, K.R., Martin, B.: A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling. In: Brusilovsky, P., Corbett, A.T., de Rosis, F. (eds.) UM 2003. LNCS, vol. 2702, pp. 313–322. Springer, Heidelberg (2003)

Murray, T.: An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art. In: Murray, T., Blessing, S., Ainsworth, S. (eds.) Authoring Tools for Advanced Learning Environments, pp. 491–544. Kluwer Academic Publishers, Dordrecht (2003)

Mayers, A., Lefebvre, B., Frasson, C.: Miace, a Human Cognitive Architecture. SIGCUE Outlook 27(2), 61–77 (2001)

Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J.: Authoring Constraint-based Tutors in ASPIRE. In: Ikeda, M., Ashley, K.D., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 41–50. Springer, Heidelberg (2006)

Najjar, M., Mayers, A., Fournier-Viger, P.: A Novel Cognitive Computational Knowledge Model for Virtual Learning Environments. The WSEAS Transactions on Advances in Engineering Education 3(4), 246–255 (2006)

Orey, M.A.: Interface Design for High Bandwidth Diagnosis: The Case of POSIT. Educational Technology 30(9), 43–48 (1990)

Orey, M.A., Burton, J.K.: POSIT: Process Oriented Subtraction-Interface for Tutoring. Journal of Artificial Intelligence in Education 1(2), 77–105 (1990)

Razzaq, L., Patvarczki, J., Almeida, S.F., Vartak, M., Feng, M., Heffernan, N.T., Koedinger, K.R.: The ASSISTment builder: Supporting the life cycle of ITS content creation. IEEE Transactions on Learning Technologies 2(2), 157–166 (2009)

Resnick, L.B.: Syntax and Semantics in Learning to Subtract. In: Moser, J. (ed.) Addition and Subtraction: A Cognitive Perspective. Lawrence Erlbaum, Hillsdale (1982)

Sleeman, D., Kelly, E.A., Martinak, R., Ward, R.D., Moore, J.L.: Studies of Diagnosis and Remediation with High School Algebra Students. Cognitive Science 13, 551–568 (1989)

VanLehn, K.: Mind Bugs: The Origin of Procedural Misconceptions. MIT Press, Cambridge (1990)

VanLehn, K.: The Behavior of Tutoring Systems. International Journal of Artificial Intelligence in Education 16(3), 227–265 (2006)

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M.: The Andes Physics Tutoring System: Lessons Learned. International Journal of Artificial Intelligence in Education 15(3), 1–47 (2005)

Wenger, E.: Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge. Morgan Kaufmann Publishers, Los Altos (1987)