INTERACTIVE LEARNING ENVIRONMENTS, 2016
http://dx.doi.org/10.1080/10494820.2015.1131167

The design and development of the dragoon intelligent tutoring system for model construction: lessons learned

Jon Wetzel, Kurt VanLehn, Pradeep Chaudhari, Avaneesh Desai, Jingxian Feng, Sachin Grover, Reid Joiner, Mackenzie Kong-Sivert, Vallabh Patade, Ritesh Samala, Megha Tiwari and Brett van de Sande

School of Computing, Informatics, and Decision Science Engineering, Arizona State University, Tempe, AZ, USA

AQ1

ABSTRACT
This paper describes *Dragoon*, a simple intelligent tutoring system which teaches the construction of models of dynamic systems. Modeling is one of eight practices dictated in two new sets of educational standards in the USA, and Dragoon is one of the first systems for teaching model construction for dynamic systems. Dragoon can be classified as a step-based tutoring system that uses example-tracing, an explicit pedagogical policy and an open learner model. Dragoon can also be used for computer-supported collaborative learning, and provides tools for classroom orchestration. This paper includes descriptions of the features, user interfaces and architecture of the system; compares and contrasts Dragoon with other intelligent tutoring systems; and presents a brief overview of formative and summative evaluations of the software in both high school and college classes. Of four summative evaluations, three found that students who used Dragoon learned more about the target system than students who did equivalent work without Dragoon.

ARTICLE HISTORY
Received 20 February 2015
Revised 8 December 2015
Accepted 9 December 2015

KEYWORDS
Intelligent tutoring system; model construction; dynamic systems; open learner model; classroom orchestration tools; learning by authoring intelligent tutoring systems

AQ2

CE: AA QA: Coll:

Introduction

This paper has two goals. The first goal is to describe a simple intelligent tutoring system, Dragoon, which teaches a complex skill: the construction of models of dynamic systems. Model construction is an important skill for high school and college students to learn. In the Common Core State Standards for Mathematics (CCSSO, 2011), a widely adopted standard for K12 schooling in the USA, there are eight practices, and one of them is modeling. Similarly, in the Next Generation Science Standards (National, 2012), another USA K12 standard, modeling is one of eight practices. Given its importance and ubiquity in the learning objectives of math and science curricula, considerable research has been done on teaching it (VanLehn, 2013). Nonetheless, it appears that Dragoon is one of the first intelligent tutoring systems for teaching construction of models of dynamic systems.

The second goal of the paper is to present an overview of the architecture of the Dragoon software. This kind of overview is possible because Dragoon has a particularly simple architecture, and it is interesting because papers on intelligent tutoring systems rarely described their systems in enough detail for replication. This has contributed to the common misconception that the technology is too complex for real-world use. In addition to this overview, the design documents and source code are available on the GitHub website at http://github.com/bvds/LaitsV3. Dragoon is an open source project under the GNU Lesser General Public Licence.[1]

CONTACT Jon Wetzel jon.wetzel@asu.edu

In the landscape of tutoring systems, Dragoon can be classified as a *step-based tutoring system* that uses *example-tracing*, an *explicit pedagogical policy* and an *open learner model*. Dragoon can also be used for *computer-supported collaborative learning* (CSCL), and provides some tools for *classroom orchestration*. The next few paragraphs introduce Dragoon by defining the italicized terms.

Dragoon is a *step-based tutoring system* for providing coached practice on solving complex, multi-step model-construction tasks. A step-based tutoring system behaves as if it has two nested loops (VanLehn, 2006).

- The *outer loop* executes once per task. It maintains an assessment of the student so that it can recommend a task for the student to do next that should maximize the student's learning and motivation. The ideal outer loop can recommend all kinds of tasks including multi-step tasks, viewing videos, reading texts or doing tasks that require only a simple answer, for example, multiple-choice questions. Learning Management Systems and similar software are beginning to acquire this capability of personalizing their recommendations for tasks. Dragoon has some outer-loop capabilities and more will be added in future work.
- The *inner loop* executes once per step. Whenever the student takes a step in solving a task, the tutor may give feedback and hints on that step. If it decides to wait until the student has finished and submitted a solution, it can still give feedback and hints on specific steps or parts of the submitted solution. Dragoon can give both immediate and delayed feedback on steps.

Dragoon is also an *example-tracing tutoring system*, which means that for every problem that the student solves, an author must provide an example solution (Koedinger, Aleven, Heffernan, McLaren, & Hockenberry, 2004). The feedback and hints given to the student are based on comparing the student's steps to steps in the author's solution.

Whether an example-tracing tutoring system deserves the title "intelligent" is contentious. Pavlik, Brawner, Olney, and Mitrovic (2013) consider a tutoring system "intelligent" if it has sophisticated computations inside it, in which case example-tracing tutoring systems would not qualify. Aleven et al. (2015) define "intelligent" as a characteristic of the observable *behavior* of the agent, citing Newell and Simon (1972) and other authors who argued with philosophers about whether any machine could be intelligent. On the Aleven et al. definition of intelligence, most example-tracing tutoring systems, and Dragoon in particular, would qualify as intelligent tutoring systems.

Like most example-tracing tutoring systems, Dragoon is also an authoring system. That is, it has two modes. When it is in *author mode*, it assumes that the user's solution is correct and merely records it. When it is in *student mode*, Dragoon assumes the user's solution may not be correct, so it compares it step-by-step to the author's solution, and may give feedback or hints to the user.

A tutoring system's decisions about whether to give feedback and hints are driven by a *pedagogical policy*. Although the pedagogical policies of many tutoring systems are hard-coded into the software, Dragoon's pedagogical policy is represented by a table that pairs conditions with responses. Although this table is currently generated by hand, it should be possible to use machine learning to improve Dragoon's policies based on the performance of students using it (Chi, VanLehn, Litman, & Jordan, 2011; Lindsey, Mozer, Huggins, & Pashler, 2013; Lopes, Clement, Roy, & Oudeyer, 2013; Rafferty, Brunskill, Griffiths, & Shafto, 2011; Whitehill, 2012).

An *open learner model* is a display that allows students to see and evaluate their progress (Bull, Dimitrova, & McCalla, 2007). Often, students are shown an indicator of their performance and progress alongside indicators for their peer's performance and progress. Considerable user-interface and social engineering has been invested in finding displays that maximize motivational benefits (Bull & Kay, 2013; Hsiao & Brusilovsky, 2011, submitted; Kay, 2001; Martinez-Maldonado, Kay, & Yacef, 2011).

In order to encourage CSCL, Dragoon allows the instructor to define small groups of students who then have access to each other's work. Each group also has a single shared piece of work that all members can edit. Each group also has their own forum, which they can use to document and discuss their work.

When students are working either alone or in small groups in a classroom, it is often difficult for the teacher to monitor their progress, detect impasses and offer guidance appropriately. This problem is often called the *orchestration* problem (Dillenbourg, Jarvela, & Fischer, 2009). Dragoon provides a common tool for orchestration, which is a teacher's dashboard that displays an overview of the class that is updated in real-time (Verbert et al., 2013). It also allows the teacher to peek at an individual students' current work. The dashboard runs on a tablet, so a teacher can carry it around the classroom.

Dragoon 2, the version of the system described here, is a complete rewrite of Dragoon 1. Prior to Dragoon 1, three versions of a system dynamics tutoring system were built as part of the Augmented Meta Tutoring project (Girard, Chavez-Echeagaray et al., 2013, Girard, Zhang et al., 2013; Gonzalez-Sanchez, Chavez-Echeagaray, VanLehn, & Burleson, 2011; VanLehn et al., 2011; Zhang et al., 2013, in press). We have learned much about how to make this class of tutoring system simple and powerful.

The remainder of this paper presents a user's eye view of the system, and then describes its architecture. A final section describes the lessons learned during its construction and our recommendations for constructing similar systems.

Related work

The main reason for developing Dragoon was to provide technology that could be used to help students learn how to construct models of dynamics systems (a cognitive skill) and then use that skill to deepen their understanding of natural and engineered systems. Perhaps the first technology with the same goal was Stella (Doerr, 1996; Richmond, 1985), which was a graphical model editor but not a tutoring system. Stella's graphical stock-and-flow notation is now widely used. It appears in university textbooks on business, ecology and many other topics. It is virtually the only notation for system dynamics used in papers published in system dynamics journals and conferences (listed at http://www.systemdynamics.org/).

Before designing Dragoon, we built a simple model editor that used stock-and-flow notation. High school students who used it for two hours reported that the stock-and-flow notation was confusing (VanLehn, 2013). They gave us many suggestions for improving the notation, which led to the notation now used by Dragoon. We see Dragoon's notation not as a replacement for the stock-and-flow notation, which is so well-entrenched that it will probably never be replaced, but as an introduction to it and perhaps even to the differential equation notation used by engineers and others.

Model-It was another model editor but not a tutoring system (Crawford & Cullin, 2004; Lee, Jonassen, & Teo, 2011; Metcalf, Krajcik, & Soloway, 2000). It also used a graphical notation for models that was simpler than the stock-and-flow notation. Ninth-grade students used it as part of an inquiry activity, wherein they studied a stream near their school and created a model of its pollutants. Many of the lessons learned during the formative evaluations of Model-It were incorporated into the design of Dragoon, such as the need for the Target Node Strategy to prevent novices from floundering. AQ3 AQ4 AQ5

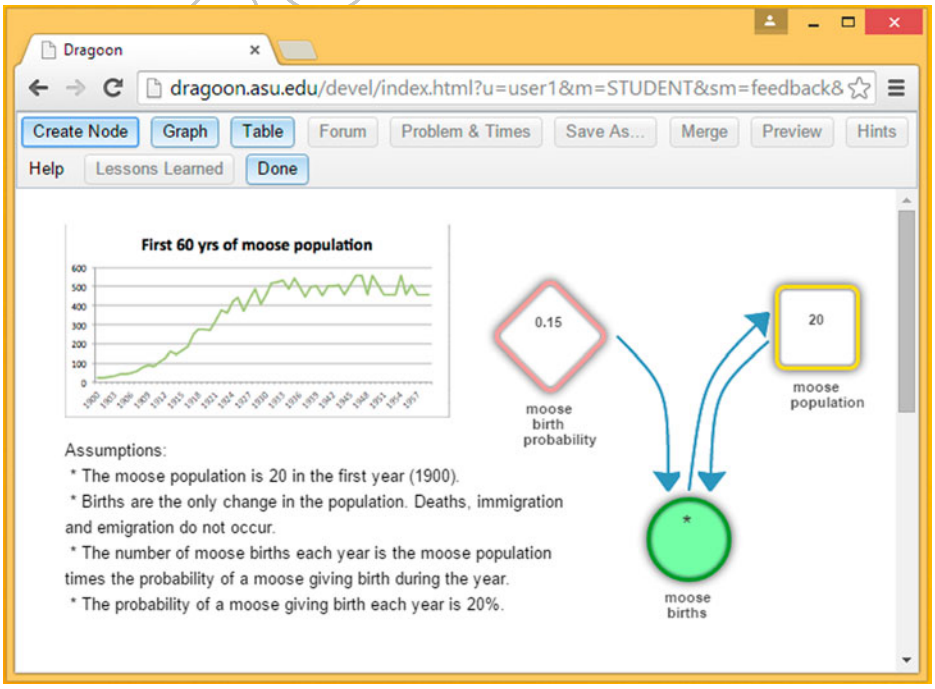
Besides Dragoon, the only tutoring system for system dynamics model construction appears to be a variant of Co-Lab (Bravo, van Joolingen, & de Jong, 2009). It used the stock-and-flow notation, and was an example-tracing tutor like Dragoon. Because the Co-Lab tutoring system was the first tutoring system for system dynamics modeling, it uncovered a problem that its predecessors did not face. All predecessor systems were merely editors, so they let students choose names for the nodes. The Co-Lab tutor did too, but then it had to match those names to the names on the nodes of the author's model. Even with considerable work, 22% of the names could not be matched automatically and had to be matched by humans. Dragoon avoids this issue by having the author specify a list of possible names. Authors can choose to add distractors to the list, namely names for quantities that are not needed in the model.

Although Stella, Model-It and Co-Lab are the systems most similar to Dragoon, ideas for its design came from a much wider class of systems. This included other editors for constructing models and tutoring systems for math and science. VanLehn (2013) reviews these systems and points out relevant ideas for Dragoon-like applications.

The behavior of dragoon

When using Dragoon, the student's task consists of reading about a dynamic system and constructing a model of it. A *system* is just a part of the real world, and a *dynamic system* is a part of the real world that changes over time. When students are just starting to learn Dragoon, systems are described with succinct, highly explicit text and a graphic, as shown in the left half of Figure 1. As students become more skilled, they are given tasks where the text is less explicit and more typical of real-world scenarios.

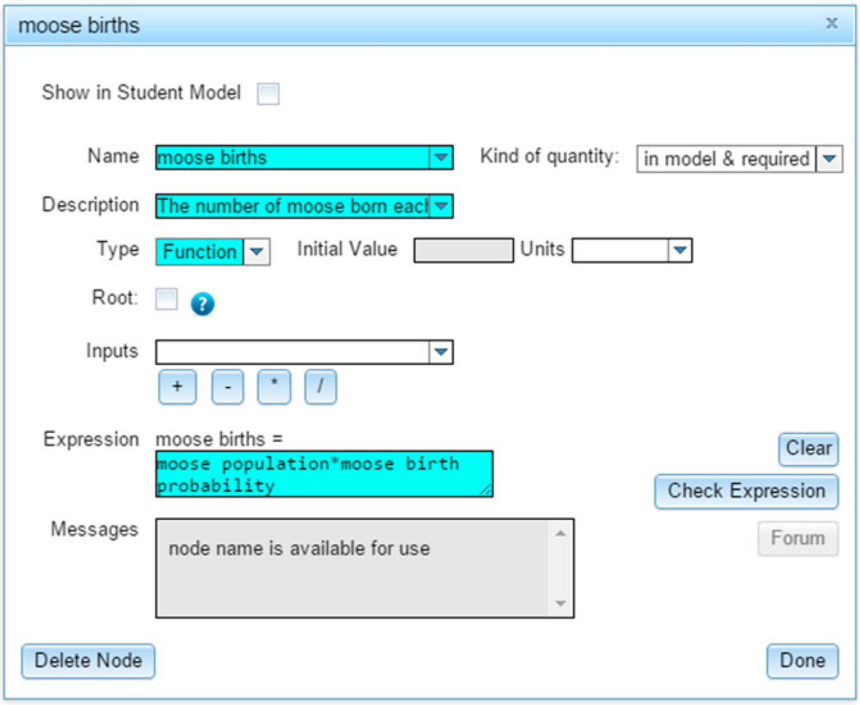
In professional applications and university engineering courses, a *model* of a dynamic system is a set of coupled first-order differential equations. However, Dragoon is aimed at high school students, and they have no knowledge of differential equations. Thus, Dragoon uses a graphical representation similar to the one pioneered by Stella (Doerr, 1996). Figure 1 shows a simple model in Dragoon. Each node in the directed graph represents both a quantity and how its value is calculated. For every input to the calculation, there is a link coming into the node from a node representing that input quantity. A square node represents an accumulator – a quantity whose value is the integral of its inputs over time. A circular node represents a function – a quantity whose value is an ordinary function (i.e. no integrals or differentials) of its inputs. A diamond-shaped node is a parameter, which is a quantity with a constant value. Users construct a model by clicking on the *Create Node* button, and then filling in a form (Figure 2) with a name for the node, a description of the quantity the node represents,



Assumptions:
* The moose population is 20 in the first year (1900).
* Births are the only change in the population. Deaths, immigration and emigration do not occur.
* The number of moose births each year is the moose population times the probability of a moose giving birth during the year.
* The probability of a moose giving birth each year is 20%.

Figure 1. A simple Dragoon model.

AQ18


moose births

Show in Student Model ☐

Name: moose births
Kind of quantity: in model & required
Description: The number of moose born eacl
Type: Function
Initial Value:
Units:
Root: ☐
Inputs:

Expression: moose births =
```
moose population*moose birth
probability
```
Messages: node name is available for use

Figure 2. The node editor in author mode.

the node's type (accumulator, function or parameter), its inputs, and how its value is calculated. When the user clicks on the Graph or Table button, Dragoon pops up plots or tables of the quantities as a function of time (see Figure 3). The sliders allow the user to temporarily change the value of the parameters and observe the resulting change in the plots and tables.

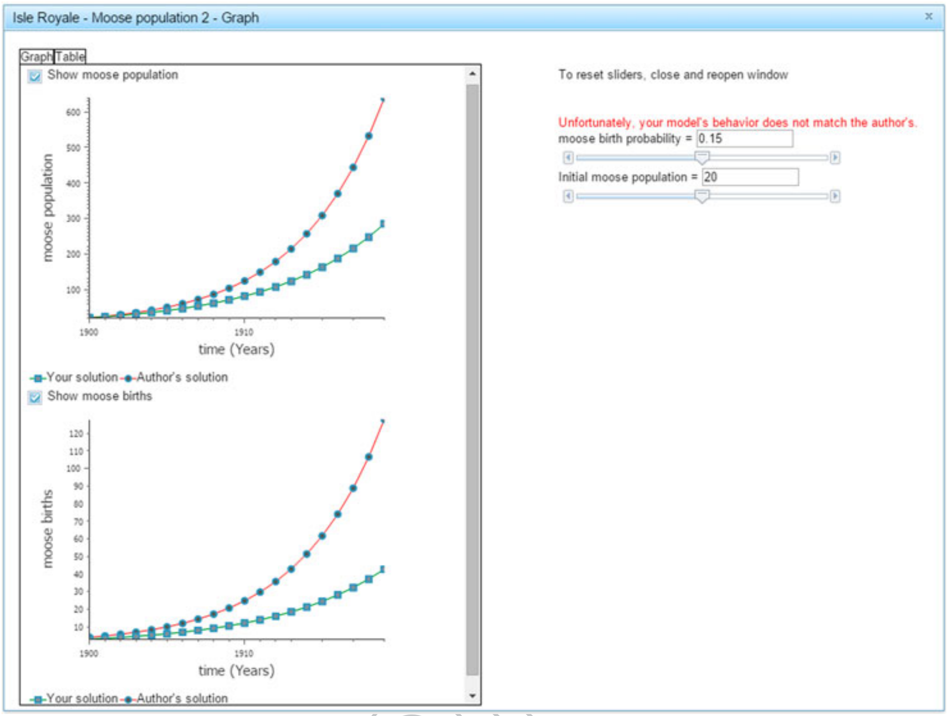
The author uses the above tools in Dragoon's *author mode* to construct a model and decide how much of it should be presented initially to the student (see check box at the top of Figure 2). Additionally, authors may use the *Merge* button to import nodes from an existing model into their own, the *Copy to …* button to create a new copy of the model to work with and the *Preview* button to see how the problems work for the student (see top of Figure 1). The *Problem & Times* button accesses editors for the rest of the information that defines a model construction problem, such as the system description's text and image (see left side of Figure 1). Except for the images, which must be imported, Dragoon is self-contained in that it is the only system required for authoring a model construction problem.

When authoring is complete, the problem may be opened by students using one of Dragoon's *student modes*. There are four student modes in Dragoon, each giving a varying amount of helpful feedback.

In *no feedback mode*, the student receives no feedback as they construct a model from the given description of the system. We have used this mode for exams and as a control condition in our experiments.

In *delayed feedback* mode, the graphs show the values of both the student's model and the author's model, as well as a message indicating whether the two models match. Typically, students are required to work on a model until the values match for every quantity. That is, when the values are graphed (see Figure 3), the red and green plots are identical.

In *immediate feedback mode*, Dragoon provides feedback on each step in the model-construction process. It colors an entry in the node editor green if its value matches the corresponding value in the



Isle Royale - Moose population 2 - Graph

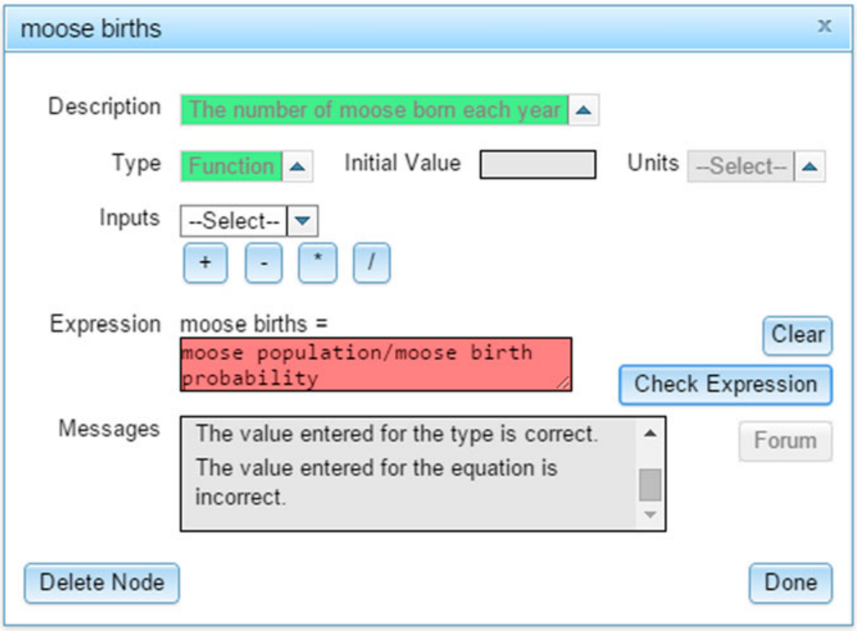
To reset sliders, close and reopen window

Unfortunately, your model's behavior does not match the author's.

moose birth probability = 0.15

Initial moose population = 20

Figure 3. Graphs of values of the non-constant variables in the model. Sliders control the values of constants.



moose births

Description: The number of moose born each year

Type: Function

Initial Value:

Units: --Select--

Inputs: --Select--

Expression: moose births =

```
moose population/moose birth
probability
```

Messages:

The value entered for the type is correct.
The value entered for the equation is incorrect.

Figure 4. The node editor in immediate feedback mode.

author’s model, and red otherwise (Figure 4). When two mismatches have been made on an entry, Dragoon fills in the entry with the author’s value and colors it yellow. Additionally, the first time a color is shown to a student, a pop-up explanation appears explaining what the color means. Finally, when the student closes the node editor, the colors of its entries are reflected in the color of its boundary (see Figure 1). The interior of a node, which is normally white, turns solid green when the student defines the whole node without making any errors. Students rapidly learn that the more green their model shows, the better. We believe this may motivate some students to try harder to suppress unintentional errors (slips) and to avoid deliberate errors, which are a kind of “gaming the system” (Baker, de Carvalho, Raspat, Corbett, & Koedinger, 2009; Hastings, Arnott-Hill, & Allbritton, 2010; Muldner, Burleson, van de Sande, & VanLehn, 2011; Roll, Aleven, McLaren, & Koedinger, 2011).

In *coached* mode, which is intended for brand new users, Dragoon gives students even more help by restricting their choices so that they flounder less. Students are guided along the *Target Node Strategy*, in which they create nodes in a logical order that has proved helpful in other tutoring systems (Chi & VanLehn, 2007a, 2007b, 2010; Zhang et al., 2014). If a student attempts to start editing a node out of order, they are directed to create the strategically preferred node or nodes first.

Students often learn more from solving a problem if they reflect on their solution and attempt to generalize from it (Katz, Allbritton, & Connelly, 2003; Katz, Connelly, & Wilson, 2007; Katz, Connelly, Wilson, & Goedde, 2006). In order to encourage reflection and generalization, Dragoon pops up a brief text when the student has finished solving a problem and viewing the graph window. The text is written by the author.

This completes the description of the inner loop of Dragoon – how it responds as the user constructs a model of a given system. Because Dragoon itself does not currently have an outer loop that intelligently selects the next problem for a student to do, we have developed several methods for using it in classes that are perhaps even better than having an outer loop. The next section describes them.

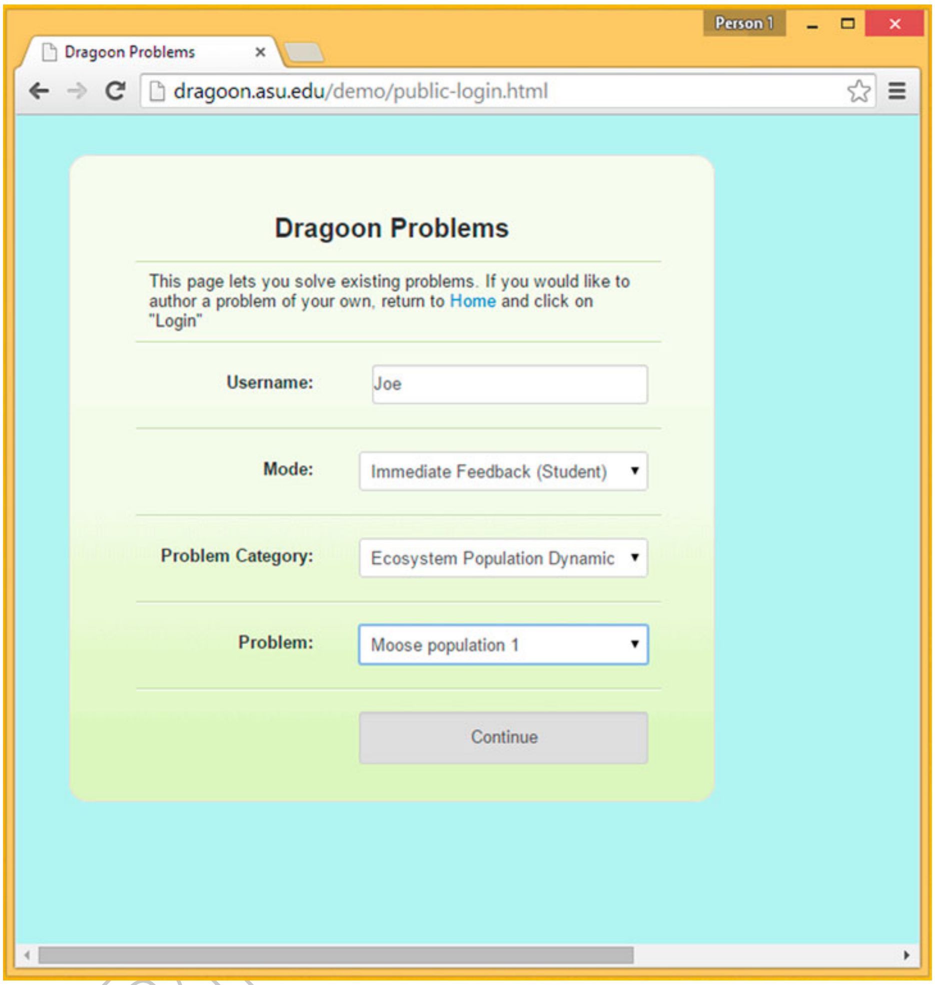
Using dragoon in the classroom

Users can select a problem using a web page (Figure 5). The web page provides access to *published* problems, which have authored models that are guaranteed to be correct representations of the given systems. Students choose a problem, a mode and what kind of user interface they want to use for entering calculations. This method of accessing problems is intended for users who are just browsing Dragoon problems. To aid in recovery when the user accidentally closes a Dragoon problem, for example, by clicking on the browser’s back button, the user can provide a username. If they come back to the Dragoon Problems page, enter their username and select the problem they were working on, then Dragoon will let them resume where they left off. However, problems done via this page may not be stored indefinitely.

For sustained usage, forum software was introduced to Dragoon as a learning content management system. Dragoon forums may be used either as a space for authors, instructors and developers to create new Dragoon content, or to manage the workflow of a classroom using Dragoon. In the latter configuration, students in a particular course may work on Dragoon problems collaboratively or independently.

Dragoon forum users must create an account. They provide a username, password and an email address which is used to recover from forgotten passwords. When a user is logged in, their Dragoon solutions are stored indefinitely, so if they access any problem that they have worked on before, they will resume where they left off. Dragoon has two types of accounts: instructor and student. Instructor accounts can only be created by the Dragoon team.

When instructors want to use Dragoon in their class, they contact the Dragoon team. The team creates an account for the instructor and a URL for the class. The instructors must require students to use that URL when they set up a Dragoon account. If the instructor plans on having students



Dragoon Problems

This page lets you solve existing problems. If you would like to author a problem of your own, return to Home and click on "Login"

Username: Joe

Mode: Immediate Feedback (Student)

Problem Category: Ecosystem Population Dynamic

Problem: Moose population 1

Figure 5. An HTML page for launching Dragoon.

work in small groups, the instructor can ask the Dragoon team to create sections within the class. Students in the same section can share their individual work and can edit a group solution to a problem.

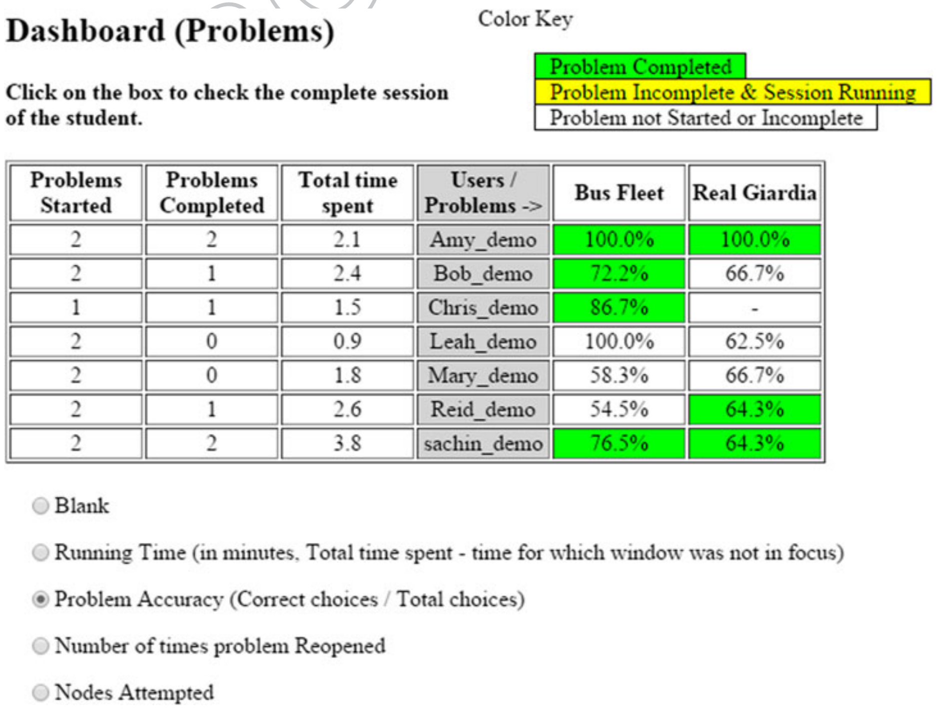
A user can also create an account from the public URL, dragoon.asu.edu. They will be placed in a giant “class” of all users who also logged in via the public URL. This allows students to use Dragoon in a sustained way even if they are not part of a formal class that is using Dragoon.

When a user is logged in, the user can author Dragoon problems as well as solve Dragoon problems authored by others. (The exception being the public forum, on which authoring requires permission from the Dragoon team.) Instructors can author problems and release them to all the students in the instructor’s class. Students can release their problems only to students in their section. (Or not at all, if the instructors ask the Dragoon team to restrict the authoring feature on their class/section forum.)

Currently, Dragoon does not have an intelligent outer loop, so users are in charge of selecting appropriate tasks for themselves. In particular, users can open any published problem as well as any problem authored for them as a member of a class or section.

In most classes so far, instructors gave students workbooks either as hardcopy or PDF. These workbooks were used to present tutorial information on how to use Dragoon and non-Dragoon questions, that build models of the water subsystem. Using the forum facilities of Dragoon, they examined and commented on each other's models. They then developed a consensus model for their subsystem. At this point, the instructor rearranged the class into jigsaw groups (Aronson, Blaney, Stephan, Sikes, & Snapp, 1978). Each jigsaw group consisted of three members, one for each subsystem. Students engaged in the LAITS activity (discussed earlier) three times. One student, let us say the one who authored a water model, tries to sit silently while the other two students solve the Dragoon problem of (re-)constructing the author's model using its forum as a guide. When both tutees have finished, the jigsaw group focuses on the next subsystem, for example, Parks. A different student now has the author role, and a different pair are tutees. The LAITS activity sometimes causes vigorous discussions when the tutees disagree with the modeling choices made by the author. We believe that LAITS causes jigsaw groups to learn much more than they would if each author simply lectured and demonstrated her or his model. After they completed the LAITS activity three times, the jigsaw groups built a model for the whole Phoenix urban ecosystem by combining Dragoon models for the three subsystems. The jigsaw groups documented their decisions, which were often quite value-laden, using the forum attached to their model. The instructors and students found this activity extremely rewarding and engaging (Iwaniec et al., 2014). This use of Dragoon illustrates how it can be used to support and perhaps enhance collaboration.

The CSCL literature suggests that instructors can benefit from real-time displays of the students' progress, as this can help them orchestrate the classes' activities (Dillenbourg & Jermann, 2010; Prieto, Dlab, Abdulwahed, & Balid, 2011). Figure 6 shows Dragoon's dashboard. The rows correspond to students and the columns on the right correspond to Dragoon problems. The color of the cells on the right indicates progress. Data inside the cell can in principle be any measure of performance. Currently, Dragoon provides both error rates and time-on-task. Clicking on a cell causes the students' problem's current state to be displayed on the teacher's screen. During class, this makes it easier for teachers to locate students who most need guidance.



Dashboard (Problems)

Color Key

🟩 Problem Completed
🟨 Problem Incomplete & Session Running
⬜ Problem not Started or Incomplete

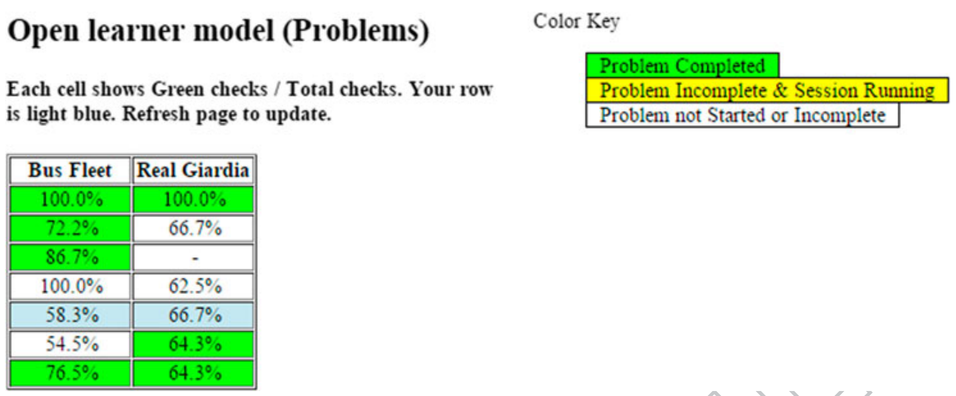
Click on the box to check the complete session of the student.

Problems Started	Problems Completed	Total time spent	Users / Problems ->	Bus Fleet	Real Giardia
2	2	2.1	Amy_demo	🟩 100.0%	🟩 100.0%
2	1	2.4	Bob_demo	🟩 72.2%	66.7%
1	1	1.5	Chris_demo	🟩 86.7%	-
2	0	0.9	Leah_demo	100.0%	62.5%
2	0	1.8	Mary_demo	58.3%	66.7%
2	1	2.6	Reid_demo	54.5%	🟩 64.3%
2	2	3.8	sachin_demo	🟩 76.5%	🟩 64.3%

○ Blank
○ Running Time (in minutes, Total time spent - time for which window was not in focus)
[✓ Problem Accuracy (Correct choices / Total choices)]
○ Number of times problem Reopened
○ Nodes Attempted

Colour online, B/W in print

Figure 6. Teacher's dashboard.


Open learner model (Problems)

Each cell shows Green checks / Total checks. Your row is light blue. Refresh page to update.

Color Key

- 🟩 Problem Completed
- 🟨 Problem Incomplete & Session Running
- ⬜ Problem not Started or Incomplete

Bus Fleet	Real Giardia
100.0%	100.0%
72.2%	66.7%
86.7%	-
100.0%	62.5%
58.3%	66.7%
54.5%	64.3%
76.5%	64.3%

Figure 7. Student's open learner model.

After class, the dashboard makes it easy for teachers to provide feedback and grades to students. The teacher can copy and paste the dashboard into a spreadsheet, allowing the teacher to aggregate performance measures in a variety of ways. Moreover, the spreadsheet cells continue to have links to the students' problems. When the teacher double-clicks on a spreadsheet cell, the browser opens the URL that accesses that the student's problem's state. The teacher can edit the students' model or leave comments in the forum for that problem. Although students can in principle print and submit their models using their browser's print command, there is no longer a need to do so. Moreover, they cannot forget to submit their homework because the teacher can always see it. Teachers no longer need to use class time to hand back marked-up copies of homework.

Figure 7 shows the open learner model, which is like the teacher's dashboard but designed for students. It hides the names and some of the data. It highlights the row that belongs to the student who is viewing it. Motivation was intended to be bolstered both by the open learner model and by the coloring of nodes in the model (green, yellow and red perimeters: a solid green interior).

In short, although Dragoon is an intelligent tutoring system with authoring capabilities, it is not being used as a tutor that paces a student through a curriculum. Instead, Dragoon is being used much like any other educational technology – as a media and guide for student activity which can be combined in innovative ways with other technology, paper and discussions by small groups and the whole class.

Dragoon's overall architecture

Dragoon is a web-based system in that the main code runs in the user's browser but is stored on a server, and that as Dragoon runs, all persistent data are stored on the server instead of the user's disk (see Figure 8). This section describes some of the details of the client and server communications and how they divide the workload of the system among themselves.

A Dragoon server consists of a web server (Apache), a forum service (phpBB) and a database (MySQL). The database contains tables for:

- The student's work on each problem (in the form of a *model* as discussed below).
- The student state (information about the student that persists and evolves across problems).
- Logs of specific student actions along with their interpretation.

The web server stores the following information:

- JavaScript code, which is downloaded to the client on demand and runs in the client's browser.
- A set of scripts that provide access to the database when called by the JavaScript code or the forum service.

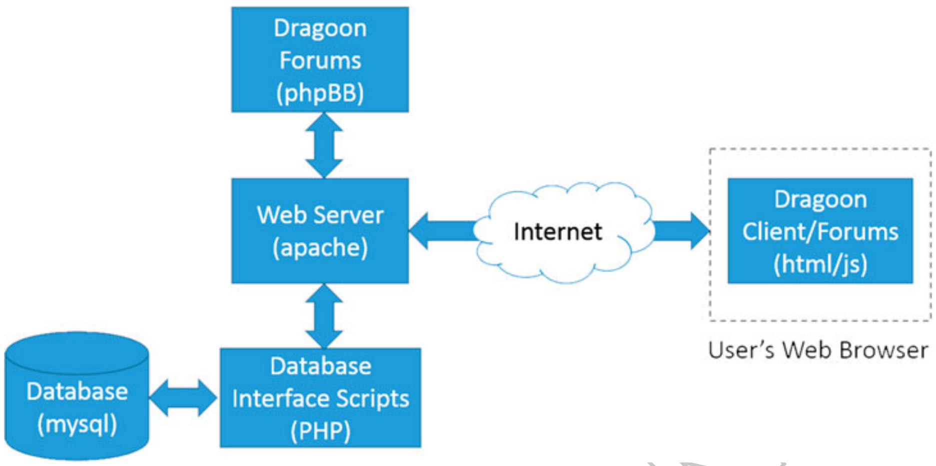


Figure 8. The overall architecture of Dragoon.

- A set of published models.
- Workbooks and other passive resources.

When a client opens a Dragoon problem, it loads or creates an object in the program called a *model*. The model object contains the following:

(1) Information that the author edits via the *Problem & Times* button: The text and image that describe the system to be modeled; the text used for reflective debriefing of the student when a problem has been solved; the start time, stop time and time step of the model.
(2) The author's model of the system, if one exists.
(3) The student's model of the system, if one exists.
(4) Historical information about the construction of this model, such as counts of the errors that the student has made so far while constructing a node.

A *published* model has only the first two items because it represents a task that the student can do. As the student works on this task, items 3 and 4 are populated. The student's model is stored in the database on the server indexed by the student name, problem name and section name. Sections are named subsets of the students in a defined class. The students in a section may be allowed to access each other's models.

The client updates the server regularly (whenever the node editor closes or a node is moved), so that little work is lost if the client stops (e.g. the student hits the back button on the browser, or closes the browser) or the network connection breaks.

In addition to regularly sending updates to the student's model in the server database, the client sends log events and changes to student state variables. The student state variables represent information that should not be stored inside a model because it evolves across models. For instance, Dragoon's current policy is to display the pop-up message "Green means that the entry is correct," only twice, as a way of teaching novices what the color-coded feedback means. However, the first and second displays of the message might occur during different problems, so a counter cannot be stored inside each model. Instead, a student state variable is used to hold the counter for the number of times that the message has been displayed.

Let us consider a typical session with Dragoon. Suppose that the student is looking at http://dragoon.asu.edu/demo/public-login.html, which currently looks like Figure 5. The user has selected a problem, "Moose population 1" of the problem set "Ecosystem Population Dynamics," which is being done in immediate feedback student mode.

When the user clicks on the Continue button, the selected information is sent to the Dragoon server as a URL (i.e. http://dragoon.asu.edu/demo/index.html?u=kvl1&m=STUDENT&sm=feedback&is=structured&p=isle1&s=public-login.html). After the requested document (index.html) is returned from the server, the client then requests the model for the selected problem. When the server gets such a request, it first looks in the database to see if this user has already worked on this problem. If so, it returns the model in whatever state it was in when the user last modified it. If the user has not worked on this problem before, then the server returns the published model for that problem and initializes a copy in the database.

The Dragoon JavaScript code merely provides event handlers for all the buttons, menus and other controls displayed on the page. The browser monitors the position of the mouse, the state of its buttons, etc. When the browser decides that the user has clicked on, say, the Create Node button, the browser calls the event handler associated with that object. The event handler is Dragoon code, which in the case of the Create Node button adds a node icon to the main window and then opens a new window for the node editor and populates it with buttons, menus, type-in boxes and other controls. It also adds a new node to the model (i.e. to the data structure we call "the model") and sends a log event to the server describing the student problem-solving step and its interpretation.

In short, Dragoon is a typical web-based system. Because it follows standard conventions, much of its functionality is provided by open-source code from the Dojo library (http://dojotoolkit.org) and jsPlumb (http://jsplumbtoolkit.com). The current Dragoon project contains about 9,191 lines of JavaScript code. The total download including libraries is currently 13.5 MB before compression.

Although Dragoon is intended to be multi-platform, updates to the libraries (e.g. Dojo, jsPlumb) tend to lag updates to the browsers and operating systems, which means that exactly which platforms and browsers it works on can vary even when the Dragoon code does not change. At this writing, Dragoon works on Google Chrome®, Mozilla®, Safari® and Internet Explorer® running on (when applicable) Windows 7®, Windows 8®, Mac OSX®, Android® and iOS®. It has been tested on laptops, tablets and a few mobile phones.

In contrast, Dragoon 1 was a Java-based web-based system with 17,219 lines of Java code and a 19 MB download *after* compression. Moreover, when we used Dragoon 1 in an Arizona State University class during the spring semester of 2014, we found that approximately 70% of the students needed to install or upgrade Java before they could start using Dragoon. The biggest stopper was an incompatibility in Java 7 with Mac OSX 10.6, which at least two students were running in the first session of the spring 2014 class. They could not run Java or Dragoon without doing a full OS upgrade first, which they could not do within the time period of the first class. About 25% of the remaining students were unable to use their browser of choice and had to switch to Chrome. Two or three students had the JNLP file type associated with programs other than Java Web Start, which required changing their browser settings, which these particular students did not know how to do. Discounting the people who could not install Java at all due to OS issues, setup took between 5 and 20 minutes depending on the user's computer proficiency and number of permissions/program associations/browser/*et cetera* issues encountered. For a class of about 40, the instructor, the TA and one of the Dragoon project developers provided tech support during the initial install process.

The conceptual architecture and algorithms of dragoon

Dragoon is essentially an editor combined with a pedagogical policy module. Let us first describe it as an editor, and then indicate how its editor architecture was modified to incorporate the pedagogical module.

Like many editors, Dragoon uses a model-view-presenter (MVP) software architectural pattern (Alles et al., 2006; Zhang & Luo, 2010). The MVP *model* consists of a data structure and access methods that represent the thing being edited (e.g. a document or drawing). The *view* is code that draws the model on the screen. The *presenter* consists of event handlers associated with a particular user interface action (e.g. selecting from a menu, clicking on a certain button or typing in a particular text box). When the user performs an action on the user interface, the associated event handler is triggered, the presenter modifies the model, and then tells the view what needs to be updated.

In the case of Dragoon, the MVP *model* is a system dynamics model. The *view* consists of the main window, which shows the nodes, links, etc., and the node editor. For the main page, there are event handlers associated with the buttons on the top menu bar, an event handler for dragging a node and a handler for clicking or double-clicking on a node. On the node editor, there are event handlers associated with each of the menus, type-in boxes and buttons.

When Dragoon is in author mode, the editor updates the model based on user actions with relatively little analysis. However, when Dragoon is in student mode, the editor compares the user's input to the author's model before responding to a user interface action. For instance, suppose the user has the node editor open on "moose births" (see Figure 4) but selects "accumulator" as the type of a node. The first thing the associated handler does is access the corresponding node "moose births" in the author's model to determine whether the student input matches the correct type. It sends this information to the pedagogical policy module, which looks up the appropriate policy and returns directives telling the presenter what to do. In Dragoon, these policies are defined by the current mode (i.e. Coached, Immediate Feedback, Delayed Feedback and Editor). For instance, if the user's selection of "accumulator" matches the type of the node in the author's model and the policy being used is in coached or immediate feedback mode, then the code tells the presenter that the type selection control should be colored green. On the other hand, if the node types do not match and the pedagogical module finds that this is the user's first error on this menu, then the pedagogical module instructs the presenter to give a hint and turn the type selector control red.

The design of most event handlers in the presenter is relatively straightforward. However, there is one handler that is more complicated. When the student types in a formula, the handler for the equation input box must determine whether the user's formula matches the corresponding formula in the author's model. This requires parsing the user's typing and informing the user of syntactic mistakes if the formula fails to parse or the variables in the equation have not been defined (possibly due to a typo). However, even when the user's formula is syntactically perfect, it is not easy to determine if it matches the author's formula. For instance, $a*(b+c)$ should match $b*a+a*c$ or even a^1(b*42 + c*42)/(84/2). Dragoon uses a *color-by-numbers* strategy to compare mathematical expressions (Shapiro, 2005; VanLehn et al., 2005). For each variable, a random number is generated and substituted for the variables' occurrences in the expressions. Then the two expressions are evaluated numerically. If the resulting numerical values for the expressions disagree, then the expressions are not equivalent. If the values agree, then there is a good chance that they are equivalent. Thus, although checking the equality of the student's step and the author's step is complicated for the equation input box's event handler, it is quite simple for all the other event handlers in the presenter.

In addition to coloring controls or giving hints, the pedagogical module can constrain what actions the student may take. For example, consider the situation where the user has just created a new node in student mode but has not yet selected a description for that node. If the user were to select a type for the node, then the pedagogical module could not tell whether the selection was correct or incorrect because it would not know which node in the author's model to compare it to. Although there are a variety of possible solutions to this problem, Dragoon's pedagogical policy is that when it is in student mode, then the user must select a description for a new node before doing anything else to it. Once the user has selected an acceptable description, the pedagogical module then instructs the presenter to enable the other input controls.

In addition to the color-based feedback and control locking, Dragoon can display hints in pop-up windows. For example, novices often start by defining a parameter for every number mentioned in the problem statement. While this works adequately on introductory problems, we believe this practice does not promote deep understanding, and it fails to prepare the student for real-world situations where there is always extra information. Thus, more advanced problems include irrelevant facts and numbers, and the list of possible names for nodes includes ones that are irrelevant to the problem solution. (The author indicates which nodes are distractors via the control labeled *Kind of quantity* in Figure 2.) If the student selects such a name, the selection turns red and the student gets the following hint:

> You tried to define a parameter for a number you read in the problem. Not all numbers in the problem statement are necessary for the model. You will save effort if you follow the Target Node Strategy, which says you should start by defining a node for a quantity that the problem asks you to graph, then define nodes for its inputs, and then define nodes for their inputs, etc. That way, every node you create is an input to some node.

This hint refers to the Target Node Strategy, which is part of the instructional materials and is often mentioned in hints. If Dragoon is in coached mode, students are required to follow the Target Node Strategy. If the student makes this same mistake later, either in this problem or a subsequent one, then Dragoon does not present the same hint because students rapidly recognize that they have seen it before and skip reading it. Instead, Dragoon presents a shorter version:

> Not every number in the problem statement is necessary for the model. You should define a node for a quantity only when either (1) it is required as input to a previously defined node, or (2) the problem statement asks you to graph it.

On the further occurrences of this error, Dragoon says:

> Please be sure you need a node before defining it. Even if a number appears in the problem statement, it may not be needed in the model.

The pedagogical module maintains a count of how often this error has occurred so that it can present the appropriate hint. This counter must be stored on the server as part of the student state so that its value persists across problems and sessions. Every hint type has a sequence of hints and a counter.

Currently, the only state that persists across problems and sessions are these hint counters, but we expect more such state variables to be needed in the future. Values for these variables are stored in the *student state* table on the Dragoon server. Whenever the pedagogical module detects a certain type of error, it updates that associated counter in the student state table. Likewise, it reads the associated counter from the student state table to determine which particular hint to give to the student. Thus the student state table acts as a model of the student. Currently, it contains just counters associated with various student actions. However, we expect it to eventually contain more sophisticated measures, such as estimates of what skills the student has mastered.

Evaluations

Dragoon has been shaped by both formative evaluations and summative ones. The formative evaluations use Dragoon 1, and took place during fall 2013 and spring 2014. They included use in three university classes (modeling once; sustainability twice) and four high school classes (biology, chemistry, physics and earth science). Log data and reports from observers were used to identify poor design elements and bugs. Discussions with teachers and students generated ideas for improvements.

During the 2014 and 2015, four summative evaluations of Dragoon were conducted. Each compared Dragoon-based instruction to baseline instruction, that is, the kind of instruction that was normally used in the class. In the high school evaluations, the students did all Dragoon work in class under the instructor's supervision, while in the university class worked both in class and at home.

The first summative evaluation (VanLehn, Wetzel, Grover, Van de Sande, & Dragoon, submitted) focused on the cognitive skill of model construction. Thirty-four students in a university class were given three weeks of practice in constructing models of a wide variety of systems. The systems were always described completely, in that students did not need to go to the literature to obtain information about the system. However, they were described as a science or math text would describe the system, which left the student to figure out what quantities were needed in the model and how to calculate each one. Despite the usually academic incentives (e.g. a quiz at the beginning of each class and an exam at the end of the three-week module), some students did very little work on Dragoon or the baseline system. When they were removed from the sample, the remaining Dragoon students scored reliability higher on the module exam than the remaining baseline students ($p < .021$), and the effect size was large ($d = 1.02$).

The second summative evaluation (VanLehn, Chung, et al., submitted) focused on teaching students some principles of physics by having them construct models of falling blocks. Fifty-five students in a high school physics class worked in pairs and used either Dragoon or paper to solve kinematics problems for one class period (55 minutes). Perhaps because the exposure time was so short or perhaps because of defects in the instructor-authored curriculum, there was no difference in the learning gains of the two groups.

The third summative evaluation (VanLehn, Chung, et al., submitted) focused on teaching students about the system that regulates weight gain and loss in the human body. Ninety-five students in a high school physiology classes used either Dragoon workbooks or paper workbooks to gradually build up a sophisticated model of the energy balancing subsystem of the human digestive system. Students worked in groups for three class periods. The Dragoon students learned reliably more than the baseline students ($p < .01$) and the effect size was moderate ($d = 0.62$).

The fourth summative evaluation (VanLehn, Chung, et al., submitted) focused on teaching students about predator-prey population dynamics. Fifty-nine students in high school biology classes studied population dynamics in a simple jigsaw activity. During the first half of the 300 minutes of instruction spread over several days, students worked individually to build a model of either the predators (lions) or the prey (zebras). During the second half, they were put into jigsaw pairs consisting of one student who had modeled the prey and one student who had modeled the predators. They then worked together to build a predator-prey model. Students who used Dragoon learned more than students who did all their work on paper. The difference in learning gains was reliable ($p < .000$) and large ($d = 1.00$).

Dragoon logged the authors' and students' actions in the user interface, allowing us to collect timing data. Details of these data for the preceding evaluations can be found in their respective papers, but in general we find that an individual model can be solved in the order of minutes. We work with instructors who choose the number of problems, often taking complexity into account, in order to create assignments of appropriate length for their classes.

The expert models used in these evaluations were authored by varying combinations of Dragoon team, our collaborating researchers at other universities and the course instructors. We allowed all collaborators access to author mode via the course forum. In the second summative evaluation, an instructor who was not a member of the Dragoon lab became familiar with Dragoon using our website and authored the models used for his unit himself. We have since written a manual for author mode to help future instructors and students learn to create their own models quickly.

However, authoring time for a model varies from under an hour to several hours, depending in part on how much literature research must be done to obtain the values for parameters. One example of a complex model requiring much research is the ecosystem of Phoenix, which students authored as the final project of their sustainability class (Figure 9 is the model from one group of students). As another example, consider the author's models used in the third summative evaluation. The model of blood glucose regulation was constructed in about 3 hours from a well-known and widely used quantitative model (Bergman, 2001; Bergman, Ider, Bowden, & Cobelli, 1979). However, the model of digestion took much longer. A nearly complete model was easily constructed in a few hours from qualitative discussions in the secondary literature. However, finding accurate values for the parameters required about 30 hours. The existing quantitative models were locked up in black-box websites used by athletes, actors and others interested in regulating their weight precisely. Parameter values were eventually estimated from the primary literature. The bottom line for authors is that if you know the parameter values already or are willing to use estimates, then

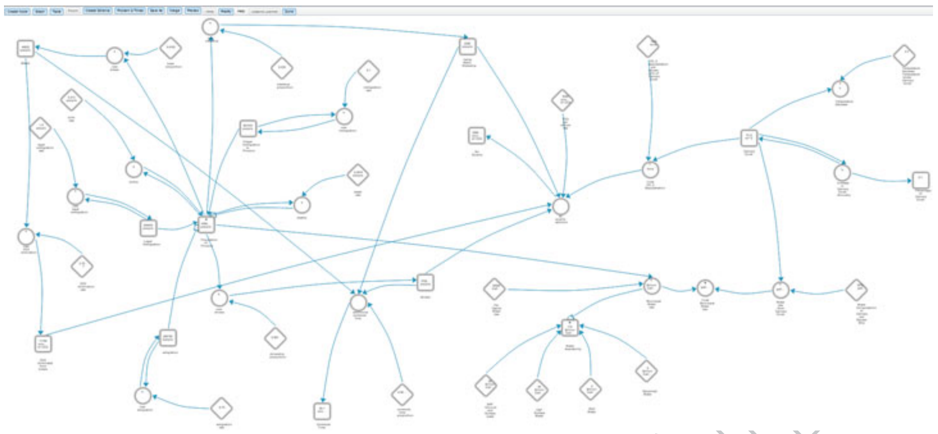


Figure 9. A model of the ecosystem of Phoenix, authored by a team of students.

authoring takes only a few hours at most, depending on your prior familiarity with the system. If you want accurate values for the parameters, then the time required to find them varies widely.

Summary and conclusions

Dragoon has a software architecture that is common among web-based applications. The tutor system itself runs entirely in the user's web browser. Static resources, such as images, text and the JavaScript code, are all stored on a web server and downloaded to the client as needed. The dynamic resources, which represent the student's state and the student's solutions to modeling problems, are stored in a database on the server. Changes to the dynamic resources that are made on the client are regularly propagated to the server, so the user never has to save anything to disk. This is a widely used software architecture, and Dragoon uses existing software libraries whenever possible.

Dragoon's conceptual architecture is a modification of a standard one, namely an MVP editor. When the browser detects a user interface action that has been registered with an event handler, it calls the associated event handler, which is part of the presenter. The presenter interprets the user's action, updates the model appropriately and tells the view how to respond. The view then modifies the user interface, perhaps giving the user a message or turning an input control red or green. When Dragoon is in author mode, it acts as a passive editor, providing little feedback to user actions. When it is in student mode, the presenter provides a much fancier interpretation of the user's action. In particular, it matches each step done by the user to the corresponding part of the author's model. This enriched interpretation is sent to the pedagogical module, which looks up the interpretation in its table of policies and retrieves a set of actions to be done by the controller. The controller implements these actions and updates the student model in an appropriate manner.

Detailed descriptions of the software architecture of Intelligent Tutoring Systems, like Dragoon, are absent from the published literature, although many man-years have been spent developing them. This description of the Dragoon architecture contributes to filling in that gap. Since many of the design issues that we discuss here should be generally applicable, we believe that a description of our software architecture will help others build similar step-based, example-tracing tutoring systems.

Dragoon is not finished. Feedback from each of the above evaluations has prompted us to significantly increase the pool of problems, develop more units for several classes (both high school and college) and add an outer loop that both assesses students' competence at system dynamics modeling and gives personalized recommendations of problems to practice on.

In conclusion, we recommend that following design process:

(1) Using the MVP architecture pattern, design an editor for the solutions that users will construct.
(2) Design the pedagogical policy table and an initial policy for it.
(3) Modify every event handler in the MVP architecture to consult the pedagogical policy before taking action. If conditions in the policy table required information about the solution state, augment the handlers to supply this information when they consult the policy.
(4) The pedagogical module should maintain a record of historical information (e.g. number of errors made by the student so far; whether a specific hint has been given) which affects its decisions.

Note

1. http://www.gnu.org/licenses/

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by National Science Foundation (IIS-1123823) and Office of Naval Research (N00014-13-C-0029).

References

Aleven, V., Sewall, J., Popescu, O., van Velsen, M., Demi, S., & Leber, B. (2015). Reflecting on twelve years of ITS authoring tools research with CTAT. In R. Sottilare, A. C. Graesser, X. Hu, & K. Brawner (Eds.), *Design recommendations for adaptive intelligent tutoring systems (Vol. III, Authoring Tools)* (pp. 263–283). Orlando, FL: US Army Research Laboratory.

Alles, M., Crosby, D., Harleton, B., Pattison, G., Erickson, C., Marsiglia, M., & Steinstra, C. (2006). *Presenter first: Organizing complex GUI applications for test-driven development*. Paper presented at the Agile Conference. AQ6

Aronson, E., Blaney, N., Stephan, C., Sikes, J., & Snapp, M. (1978). *The jigsaw classroom*. Beverly Hills, CA: Sage.

Baker, R. S. J. d., de Carvalho, A. M. J. A., Raspat, J., Corbett, A., & Koedinger, K. R. (2009). *Educational software features that encourage or discourage "gaming the system"*. Proceedings of the 14th international conference on artificial intelligence in education (pp. 475–482). Amsterdam: IOS Press.

Bergman, R. N. (2001). The minimal model of glucose regulation: A biography. In J. Novotny, M. Green, & R. Boston (Eds.), *Mathematical modeling in nutrition and health*. Amsterdam: Kluwer Academic/Plenum. AQ7

Bergman, R. N., Ider, Y. Z., Bowden, C. R., & Cobelli, C. (1979). Quantitative estimation of insulin sensitivity. *American Journal of Physiology-Endocrinology and Metabolism*, *236*(6), E667–E677. Retrieved from https://scholar.google.com/citations?view_op=view_citation&hl=en&user=BnkxW2cAAAAJ&citation_for_view=BnkxW2cAAAAJ:PyEswDtIyv0C

Bravo, C., van Joolingen, W. R., & de Jong, T. (2009). Using co-lab to build system dynamics models: Students' actions and on-line tutorial advice. *Computer and Education*, *53*, 243–251.

Bull, S., Dimitrova, V., & McCalla, G. (2007). Open learner models: Research questions (Preface to special issue of IJAIED, Part 1). *International Journal of Artificial Intelligence and Education*, *17*(2), 89–120.

Bull, S., & Kay, J. (2013). Open learner models as drivers for metacognitive processes. In *International handbook of metacognition and learning technolologies* (pp. 349–365). New York, NY: Springer.

CCSSO. (2011). *The common core state standards for mathematics*, Retrieved October 31, 2011 from www.corestandards.org..

Chi, M., & VanLehn, K. (2007a). Accelerated future learning via explicit instruction of a problem solving strategy. In K. R. Koedinger, R. Luckin, & J. Greer (Eds.), *Artificial intelligence in education* (pp. 409–416). Berlin: Springer.

Chi, M., & VanLehn, K. (2007b). The impact of explicit strategy instruction on problem-solving behaviors across intelligent tutoring systems. In D. S. McNamara & G. Trafton (Eds.), *Proceedings of the 29th conference of the cognitive science society* (pp. 167–172). Mahway, NJ: Erlbaum.

Chi, M., & VanLehn, K. (2010). Meta-cognitive strategy instruction in intelligent tutoring systems: How, when and why. *Journal of Educational Technology and Society*, *13*(1), 25–39.

Chi, M., VanLehn, K., Litman, D., & Jordan, P. (2011). Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, *21*(1–2), 99–135.

Chi, M. T. H. (2009). Active-Constructive-Interactive: A conceptual framework for differentiating learning activities. *Topics in Cognitive Science*, *1*, 73–105.

Dillenbourg, P., Jarvela, S., & Fischer, F. (2009). The evolution of research on computer-supported collaborative learning: From design to orchestration. In *Technology-enhanced learning* (pp. 3–19). Ambsterdam: Springer. AQ8

Dillenbourg, P., & Jermann, P. (2010). Technology for classroom orchestration. In M. S. Khine & I. M. Saleh (Eds.), *New science of learning: Cognition, computers and collaboration in education*. New York: Springer. AQ9

Doerr, H. M. (1996). Stella ten-years later: A review of the literature. *International Journal of Computers for Mathematical Learning*, *1*, 201–224.

Girard, S., Chavez-Echeagaray, M.-E., Gonzalez-Sanchez, J., Hidalgo Pontet, Y., Zhang, L., Burleson, W., & Vanlehn, K. (2013). Defining the behavior of an affective learning companion in the affective meta-tutor project. In H. C. Lane, K. Yacef, J. Mostow, & P. I. Pavlik (Eds.), *Artificial Intelligence in Education: 16th International Conference, AIED 2013* (pp. 21–30). Heidelberg: Springer.

Girard, S., Zhang, L., Hidalgo Pontet, Y., VanLehn, K., Burleson, W., Chavez Echeagary, M. E., & Gonzales Sanchez, J. (2013). Using HCI task modeling techniques to define how deeply students model. In H. C. Lane, K. Yacef, J. Mostow, & P. I. Pavlik (Eds.), *Artificial intelligence in education: 16th international conference, AIED 2013* (pp. 766–769). Heidelberg: Springer.

Gonzalez-Sanchez, J., Chavez-Echeagaray, M.-E., VanLehn, K., & Burleson, W. (2011). *From behavioral description to a pattern-based model for intelligent tutoring systems*. Paper presented at the Proceedings of the 18th International conference onPattern Languages of Programs (PLoP), Portland, OR.

Hastings, P., Arnott-Hill, E., & Allbritton, D. (2010). Squeezing out gaming behavior in a dialog-based ITS. In V. Aleven, H. Kay, & J. Mostow (Eds.), *Intelligent tutoring systems 2010* (pp. 204–213). Berlin: Springer-Verlag.

Hsiao, I.-H., & Brusilovsky, P. (2011). The role of community feedback in the student example authoring process: An evaluaiton of AnnotEx. *British Journal of Educational Technology*, *42*(3), 482–499.

Hsiao, I.-H., & Brusilovsky, P. (submitted). *Social progress visualization: On the crossroads of learning analytics and open student modeling*. AQ10

Iwaniec, D. M., Childers, D. L., VanLehn, K., & Wiek, A. (2014). Studying, teaching and applying sustainabilty visions using systems modeling. *Sustainability*, *6*(7), 4452–4469. doi:10.3390/su6074452

Katz, S., Allbritton, D., & Connelly, J. (2003). Going beyond the problem given: How human tutors use post-solution discussions to support transfer. *International Journal of Artificial Intelligence in Education*, *13*, 79–116. Retrieved from http://www.cogs.susx.ac.uk/ijaied/members02/archive/Vol_13/katzetal/paper.pdf.

Katz, S., Connelly, J., & Wilson, C. (2007). Out of the lab and into the classroom: An evaluation of reflective dialogue in Andes. In R. Luckin & K. R. Koedinger (Eds.), *Proceedings of AI in education* (pp. 425–432). Amsterdam: IOS Press.

Katz, S., Connelly, J., Wilson, C., & Goedde, C. (2006). *Post-practice dialogues in an intelligent tutoring system for college-level physics*. Paper presented at the American Association of Physics Teachers Summer Meeting, Syracuse, NY.

Kay, J. (2001). Learner control. *User Modeling and User-Adapted Interaction*, *11*(1), 111–127. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.214.8072&rep=rep1&type=pdf

Koedinger, K. R., Aleven, V., Heffernan, N. T., McLaren, B., & Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In J. C. Lester, R. M. Vicari, & F. Paraguaca (Eds.), *Intelligent tutoring systems: 7th international conference, ITS 2004* (pp. 162–174). Berlin: Springer.

Lindsey, R. V., Mozer, M. C., Huggins, W. J., & Pashler, H. (2013). *Optimizing instructional policies*. Paper presented at the Advances in Neural Information Processing Systems 26 (NIPS 2013). AQ11

Lopes, M., Clement, B., Roy, D., & Oudeyer, P.-Y. (2013). *Multi-armed bandits for intelligent tutoring systems. arXiv preprint*. AQ12

Martinez-Maldonado, R., Kay, J., & Yacef, K. (2011). *Visualisations for longitudinal participation, contribution and progress of a collaborative task at the tabletop*. Paper presented at the Computer Supported Collaborative Learning.

Muldner, K., Burleson, W., van de Sande, B., & VanLehn, K. (2011). An analysis of students' gaming behaviors in an intelligent tutoring system: Predictors and impacts. *User Modeling and User-Adapted Interaction*, *21*(1–2), 99–135.

National, R. C. (2012). *A Framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. Washington, DC: National Academies Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentrice-Hall.

Pavlik, P. I., Brawner, K., Olney, A., & Mitrovic, A. (2013). A review of student models used in intelligent tutoring systems. In R. Sottilare, A. C. Graesser, X. Hu, & H. Holden (Eds.), *Design recommendations for adaptive intelligent tutoring systems (Vol. I, Learning Modeling)* (pp. 39–68). Orlando, FL: Army Research Laboratory.

Prieto, L. P., Dlab, M. H., Abdulwahed, M., & Balid, W. (2011). Orchestrating technology enhanced learning: A literature review and conceptual framework. *International Journal of Technology Enhanced Learning*, *3*(6), 583–598.

Rafferty, A. N., Brunskill, E., Griffiths, T. L., & Shafto, P. (2011). Faster teaching by POMDP planning. In G. Biswas (Ed.), *Artificial Intelligence in Education* (pp. 280–287). Berlin: Springer-Verlag.

Richmond, B. M. (1985). *STELLA: Software for bringing system dynamics modeling to the other 98%*. Paper presented at the Proceedings of the 1985 International Conference of the System Dynamics Society: 1985 International System Dynamics Conference.

Roll, I., Aleven, V., McLaren, B., & Koedinger, K. R. (2011). Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction*, 267–280. AQ13

Shapiro, J. A. (2005). Algebra subsystem for an intelligent tutoring system. *International Journal of Artificial Intelligence in Education*, *15*(3), 205–228.

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence and Education*, *16*, 227–265.

VanLehn, K. (2013). Model construction as a learning activity: A design space and review. *Interactive Learning Environments*, *21*(4), 371–413.

VanLehn, K., Burleson, W., Chavez-Echeagaray, M.-E., Christopherson, R., Gonzalez-Sanchez, J., Hastings, J., … Zhang, L. (2011). *The affective meta-tutoring project: How to motivate students to use effective meta-cognitive strategies*. Paper presented at the 19th International Conference on Computers in Education, Chiang Mai, Thailand.

VanLehn, K., Chung, G., Grover, S., Madni, A., Wetzel, J., & Dragoon, t. t. (submitted). Learning about dynamic systems and domain principles: The effectiveness of the Dragoon intelligent tutoring system. *International Journal of Artificial Intelligence in Education*. AQ14

VanLehn, K., Lynch, C., Schultz, K., Shapiro, J. A., Shelby, R. H., Taylor, L., … Wintersgill, M. C. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence and Education*, *15*(3), 147–204.

VanLehn, K., Wetzel, J., Grover, S., Van de Sande, B., & Dragoon, t. t. (submitted). Learning how to construct models of dynamic systems: The effectiveness of the Dragoon intelligent tutoring system. *IEEE Transactions on Learning Technologies*. AQ15

Verbert, K., Govaerts, S., Duval, E., Santos, J. L., Assche, F. V., Parra, G., & Klerkx, J. (2013). Learning dashboards: An overview and future research opportunities. *Personal and Ubiquitous Computing*. AQ16

Whitehill, J. (2012). *A stochastic optimal control perspective on affect-sensitive teaching*. (Ph. D.), University of California, San Diego.

Zhang, L., Burleson, W., Chavez-Echeagaray, M.-E., Girard, S., Gonzalez-Sanchez, J., Hidalgo Pontet, Y., & VanLehn, K. (2013). Evaluation of a meta-tutor for constructing models of dynamic systems. In H. C. Lane, K. Yacef, J. Mostow, & P. I. Pavlik (Eds.), *Artificial intelligence in education: 16th International conference, AIED 2013* (pp. 666–669). Berlin: Springer.

Zhang, L., Vanlehn, K., Girard, S., Burleson, W., Chavez-Echeagaray, M.-E., Gonzalez-Sanchez, J., & Hidalgo Pontet, Y. (2014). Evaluation of a meta-tutor for constructing models of dynamic systems. *Computers & Education*, *75*, 196–217.

Zhang, L., Vanlehn, K., Girard, S., Burleson, W., Chavez-Echeagaray, M.-E., Gonzalez-Sanchez, J., & Hidalgo Pontet, Y. (in press). Evaluation of a meta-tutor for constructing models of dynamic systems. *Computers & Education*. AQ17

Zhang, Y., & Luo, Y. (2010). *An architecture and implement model for model-view-presenter pattern*. Paper presented at the computer science and information technology (ICCSIT), 3rd IEEE International Conference.