Opening up an Intelligent Tutoring System Development Environment for Extensible Student Modeling

Kenneth Holstein, Zac Yu, Jonathan Sewall, Octav Popescu, Bruce M. McLaren, and Vincent Aleven

Carnegie Mellon University, Pittsburgh, PA 15217, USA
{kjholste, zacy1, sewall, bmclaren, aleven}@cs.cmu.edu, octav@cmu.edu

Abstract. ITS authoring tools make creating intelligent tutoring systems more cost effective, but few authoring tools make it easy to flexibly incorporate an open-ended range of student modeling methods and learning analytics tools. To support a *cumulative* science of student modeling and enhance the impact of real-world tutoring systems, it is critical to extend ITS authoring tools so they easily accommodate novel student modeling methods. We report on extensions to the *CTAT/Tutorshop* architecture to support a plug-in approach to extensible student modeling, which gives an author full control over the content of the student model. The extensions enhance the range of adaptive tutoring behaviors that can be authored and support building external, student- or teacher-facing real-time analytics tools. The contributions of this work are: (1) an open architecture to support the plugging in, sharing, re-mixing, and use of advanced student modeling techniques, ITSs, and dashboards; and (2) case studies illustrating diverse ways authors have used the architecture.

Keywords: Authoring tools, architectures, closing the loop, student modeling, learning analytics, intelligent tutoring systems

1 Introduction

Over the last few decades, authoring tools have made the development of intelligent tutoring systems (ITSs) substantially more cost effective [1, 5, 27, 35]. Yet these tools are not always geared towards easily and flexibly accommodating advances in student modeling, which may limit the degree to which they drive innovation in ITS research and the degree to which advances in student modeling spread across ITSs. Student models have long been (and remain) a key element of ITSs. They track many pedagogically-relevant features of student learning and behavior, including the moment-by-moment development of student knowledge (e.g., [8, 11, 23, 43]), metacognitive skills (e.g., [3]), affect (e.g., [10, 14, 25]), and motivation (e.g., [4]). They are a foundation for adaptive tutoring behaviors in ITSs [11], which in turn can lead to more effective instruction [3, 4, 10, 19, 26]. Student models, and learning analytics more broadly, are also increasingly being used in tools such as dashboards, open learner

models, and classroom orchestration tools, where they can augment the perceptions of teachers [19, 42] and learners [7, 26].

However, various factors work against novel student modeling methods spreading widely in ITSs. These methods (e.g., [11, 23, 32, 43]) are often developed and tested on historical log data from educational software (i.e., "offline"). They are not commonly implemented or evaluated in real-world educational technologies, as we saw for example with AFM [6], PFA [34], and various innovative extensions to Bayesian Knowledge Tracing (BKT) (e.g., [23, 43]; but see [8]). Even when an advance in student modeling has been demonstrated in a live tutoring system, it often stays confined to that system, without being taken up in other systems (e.g., [3, 4, 10, 16]).

ITS authoring tools, and the ITS architectures with which they are integrated, could help address these challenges if they provided support for easy integration of a wide and open range of student modeling methods and analytics. Given that for many ITS authoring tools, many classroom-proven tutors exist, such authoring tool functionality could facilitate testing the *generality* of new student modeling methods across a range of tutors. Further, easy integration could facilitate further experimentation with new student modeling methods, beyond the initial offline testing, regarding how best to use these methods to enhance an ITS's functionality (e.g., with new adaptive tutoring behaviors or external learning analytics tools). Eventually, researchers may conduct more close-the-loop studies, in which the effects of new student modeling methods and analytics are rigorously tested in "live" tutoring systems (e.g., [3, 4, 7]). Results from such studies could accelerate a cumulative science of student modeling, as well as extend student modeling advances into working ITSs and educational practice.

However, ITS authoring tools rarely support *extensible* student modeling. For example, prior to the work reported in the current paper, *CTAT/Tutorshop*, an authoring environment for cognitive tutors and example-tracing tutors that has been used to build many dozens of ITSs [1], supported only student models comprising a set of BKT mastery probabilities for knowledge components (KCs) within the authored tutors. An author could not add other types of variables to the student model (e.g., to track the student's affective or motivational state, or metacognition) or easily experiment with different methods for updating or using the student model. Similarly, *ASSISTments Builder* [35] and *ASPIRE* [29], other major ITS authoring tools, do not support easy extension of their student models with new types of variables. By contrast, *GIFT* [37] does support an extensible student model based on multiple data sources (e.g., sensor data) with different time scales and granularity. Yet *GIFT* has been designed with a different focus than *CTAT*, and thus has other limitations [15]. For example, unlike *CTAT*, *GIFT* does not support non-programmer authoring of tutors with their own tutor interface and an extended step loop. We see these related, somewhat divergent, efforts as synergistic and a useful point of reference.

To address this challenge, we have extended *CTAT/Tutorshop* so authors can easily plug in an open-ended range of student modeling techniques. The extensions also support the authoring of an open-ended range of adaptive tutoring behaviors and facilitate the development of an open-ended range of student-facing and teacher-facing

support tools, including real-time tools for awareness and orchestration [7, 17]. We refer to the new architecture as the *CTAT/TutorShop Analytics (CT+A)* architecture. We aim to lower the barriers to the sharing, re-use, and re-mixing of advanced student modeling methods across researchers and research groups, with the goal of accelerating progress within a *cumulative* science of student modeling (c.f., [11, 32, 37]).

2 The CTAT/Tutorshop Analytics (CT+A) Architecture

2.1 Overview

CTAT is a widely used ITS authoring tool that supports both a non-programmer approach (example-tracing tutors) and an AI-programming approach (Cognitive Tutors, a form of model-tracing tutors) to tutor authoring. *TutorShop* is a learning management system (LMS) built for classroom use of *CTAT* tutors. Use of *CTAT* has been estimated to make ITS development 4-8 times as cost effective, compared to historic estimates of development time [1]. As evidence that *CTAT* and *Tutorshop* are robust and mature, *CTAT* has been used by more than 750 authors. Dozens of tutors built with *CTAT* have been used in real educational settings [1]. As of 2015, *CTAT*-built tutors had been used by 44,000 students, with roughly 48,000,000 student/tutor transactions, for a total of 62,000 hours of student work. Since then, there has been substantial additional use.

We first describe key elements of the *CT+A* architecture (shown in Figure 1) that existed prior to adding the new support for extensible student modeling. At a functional level, each tutor created in this architecture comprises a "step loop" nested within a "task loop" [39, 40]. The step loop supports within-problem tutoring, the task loop supports problem selection. The step loop has two key components, namely, a tutor interface and a tutor engine, both running on the client (i.e., the student's machine). The interface is where the student-tutor interactions happen; it is custom-designed for each problem type. The tutor engine interprets student actions and decides what feedback or hints to give, employing either the model-tracing or example-tracing algorithm, depending on the tutor type. The tutor's task loop is implemented in *TutorShop* and runs on the server. *CT+A* offers various problem selection algorithms that can be used within a tutor, including individualized mastery learning [8]. This method relies on a student model that, as mentioned, contains estimates of the probability that the student has mastered each of a set of KCs targeted in the current tutor unit, computed (by the tutor engine, as part of the step loop) according to a standard BKT model [8]. *TutorShop* takes care of permanent storage of the student model. It also provides learning management functionality for teachers (e.g., managing student accounts and assignments), as well as content management (e.g., it stores tutor curriculum content). This architecture has been used to build many tutors, but it cannot easily accommodate new student modeling methods. To address this limitation, we added the following key extensions:

1. An *extensible student model*. An author can now add new variables to the student model.

2. An API and template for automated plug-in *detectors* for any new student modeling variables (i.e., computational processes – oftentimes machine-learned – that track psychological and behavioral states of learners based on the transaction stream with the ITS). For the time being, we focus on sensor-free detection of student modeling variables. We have started to create a library of compatible detectors [9], so as to facilitate sharing, re-use, and re-mixing of plug-in detectors among authors;
3. Multiple mechanisms by which authors can craft tutor behavior that adapts to student model extensions, in the tutor's step loop and task loop, and
4. A *forwarding* mechanism within *TutorShop* that allows authors to pass student models to web-connected learning analytics displays on a broad range of platforms (from browser-based dashboards to wearable devices).
5. The beginnings of a library of "dashlets," to facilitate building learning analytics tools. Dashlets are re-usable interface components that can be associated with sets of analytics and configured to visualize these analytics.

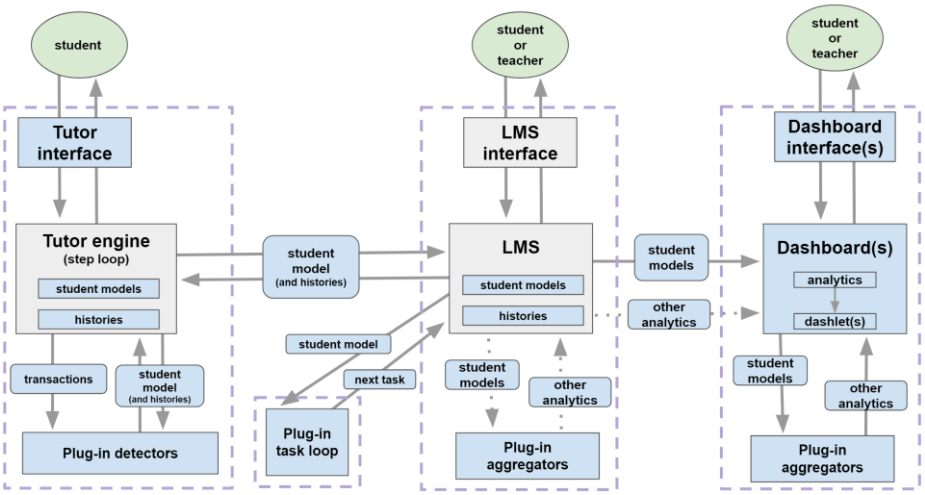

Fig. 1. Overview of the modular *CT+A* architecture, illustrating the flow of information between architectural components, with the top level (ovals) representing users. Components within dotted-line regions run on the same machine. Items in blue represent configurable components. Rounded boxes indicate information being passed between architectural components. Dotted arrows represent pathways that are not presently implemented.

2.2 The Extensible Student Model

Whereas previously the student model of a tutor built in *CTAT/Tutorshop* comprised only a set of KC probabilities, the student model is now extensible, with authors having full control over the set of variables it contains. An author can add any number of variables to the student model that capture student behaviors and inferred psychological states (e.g., knowledge, metacognitive, affective, or motivational states) [11]. The

KC probabilities remain as variables in the student model if the author so wishes. With the exception of these KC probabilities, *TutorShop* is oblivious to the semantics of the analytics in the student model (i.e., it does not have any built-in functionality that responds to the student model analytics; all such functionality must be provided by the author). A key advantage of this "semantic ignorance" is flexibility and control on the part of authors defining and using these analytics. Transparent to the author, the *CT+A* architecture maintains, in real time, two up-to-the-second copies of the student model, one within the tutor engine, one within *TutorShop*. Within the tutor engine, the student model can support adaptive tutoring behaviors. Within *TutorShop*, it can support external real-time support tools an author may wish to create or hook in (e.g., a real-time dashboard). The copy of the student model stored by *TutorShop* is kept in between problems and student sessions and is sent back to the tutor engine at the beginning of each problem/session, again transparent to the author.

2.3 Plug-in Detectors

To extend the student model, an author needs to provide automated *detectors* for all new student model variables, that is, code that computes these variables. Tutor authors can write plug-in detectors in Javascript, working from either previously-created detectors or from a generic template, available in a central, open source code repository [9]. The template defines a small number of code modules that each detector should have, namely, student model variable computations, internal feature computations, and trigger conditions for each.

To support a "remix" approach to student modeling, we have started a library of detectors that conform to this template. The library is freely available [9], and we hope it will continue to grow through community authoring. Many of the detectors currently available have been used in running ITSs and dashboards, including: multiple variants of the Help Model [3], BKT [8], various moving average detectors [34], and detectors of unproductive persistence or "wheel-spinning" [21]. Paquette et al. have also recently developed and shared a detector of "gaming the system" behavior in ITSs [4] that generalizes well across a diverse range of systems [32].

Detectors in *CT+A* are plug-in agents that rely on three sources of input. First, they listen to the transaction stream coming from the tutor engine; each transaction describes a student action, such as an attempt at solving a step or a hint request, as well as the tutor's response, such as whether the student action was correct and what KCs were involved. Each detector also listens for updates to the extensible student model (i.e., updates made by other detectors), and has access to all student model variables, in addition to any intermediate variables that the detector itself maintains (see below). Based on these inputs, each detector responds with newly computed values for its targeted student model variables. As a result, both copies of the student model (the one within the tutor engine and the one within *TutorShop*) are updated, transparent to the author. Student model updates are sent to *TutorShop* in a fine-grained, transaction-based message format we have adopted, a subset of *LearnSphere*'s Tutor Message format [15, 36].

Each detector can maintain an internal state in the form of a set of intermediate variables specified to conform to the detector template. Intermediate variables are not considered to be part of the student model and are therefore not accessible to other architectural components such as other detectors or aggregators. They are, however, sent to *TutorShop*, so that *TutorShop* can save a (compact) "history" for each detector. These detector histories are sent back to the tutor engine at the beginning of each problem, so that the previous state of each associated detector can be restored.

Although *CTAT* detectors typically run in live tutoring systems, they can also be used, without modifications, for offline data analyses (e.g., [19, 32]). *LearnSphere* [38], a large online data repository with many analysis tools, provides a workflow component for *CTAT* detectors, in the *Tigris* visual workflow tool, that enables running detectors against historical log data (from the same or other *CTAT* tutors).

2.4 Extended Support for Authoring Adaptive Tutor Behaviors

To enable the authoring of a wide and open range of adaptive tutor behaviors, we added two mechanisms to *CTAT* by which an author can make a tutor's behavior in the *step loop* (i.e., the within-problem tutoring support it offers [2, 40]) contingent on the extensible student model. We also made provisions for plugging in new task selection algorithms in the tutor's *task loop*.

As a first mechanism for creating step-loop tutor behaviors that are responsive to the extensible student model, authors of example-tracing tutors can use Excel-like formulas that reference student model variables. The use of formulas, attached to the tutor's behavior graph, has long been part of CTAT [1]; what's new is that formulas can reference variables in the extensible student model (see Figure 2). Formulas can affect many aspects of tutor behavior, including how the tutor interprets a student's problem-solving behavior against a behavior graph, the content of feedback and hints, and tutor-performed actions. Using these building blocks, an author can craft a wide and open-ended range of adaptive tutor behaviors, for example, presenting abstract hints to advanced students, presenting empathic hints to frustrated students, presenting unmastered steps as worked-out steps to be explained by the student, and having the tutor perform highly mastered steps for the student to reduce "busy work." Authors of rule-based tutors can also craft rules that support adaptive behaviors, taking of advantage of the extensible student model's availability in working memory.

A second mechanism addresses a limitation of the first, namely, that it cannot be used to craft adaptive tutor behaviors that respond to the very last (i.e., the most recent) student action – it lags by one student action. Sometimes, tutor behaviors are needed that are contingent upon updates of the extensible student model triggered by the very last student action. Our second mechanism lets author craft such tutor behavior, although to do so, the author must write Javascript code. Specifically, all tutors have a dedicated plug-in agent called the "Tutor's Ear", that continuously listens for updates to the student model. The Tutor's Ear has unique access to the tutor engine, meaning that it can directly trigger tutor responses. Authors can customize this detector by specifying (in Javascript code) conditions involving one or more student model variables under which a particular tutor response should be triggered. Authors can

then specify desired response actions (e.g., "ShowMessage ('Try explaining to yourself what needs to be done on this step')"), via a simple API. Ideally, *CTAT* would have a single mechanism for step-loop adaptivity based on student model variables, but a substantial re-architecting would be necessary to merge the two mechanisms.

In addition to supporting the authoring of adaptive behaviors in the tutor's step loop, we support the plugging-in of adaptive task selection methods (i.e., plug-in task loops), by making the student model available to external task selection processes.

2.5 Support for Using Learning Analytics in External Support Tools

Finally, authors may use *Tutorshop* to forward student models to web-connected learning analytics displays on a range of platforms, from browser-based dashboards to wearable devices [17]. While detectors in *CT+A* operate client-side, within individual students' tutors, and thus can only compute analytics for individual students, it is often useful for learning analytics applications (e.g., teacher dashboards) to compute analytics at higher units of analysis, such as groups of students or whole classes. For example, in classrooms in which students work with *CTAT* tutors collaboratively (c.f., [31]), information about the relative performance and contributions of the students in a group might be useful to display to teachers. To address this need, the extended architecture provides an "aggregator" API to enable authors to compute custom group- or class-level analytics from student model variables across multiple students.

Authors of learning analytics tools can write custom "aggregators" in JavaScript to calculate new values from detector analytics across specified sets of students. Aggregator calculations can be triggered by incoming student model updates. We created the *Aggregator House* (AggHouse), a JavaScript/Node.js [30] library that can invoke aggregators either on the *Tutorshop* server, or directly on a dashboard client. Results from aggregators can, in turn, be used to update real-time dashboard displays.

To facilitate building analytics tools that can be used in conjunction with *CTAT*-built tutors (e.g., dashboards and orchestration tools), we provide an API called the *CT+A Live Dashboard*, which includes the beginnings of a library of "dashlets," interface components for analytics tools. Authors may use the built-in dashlet components or create new dashlets (using Javascript). In addition to supporting the building and deployment of web browser-based dashboards, *Tutorshop* can also forward analytics to tools running on external hosts, via a real-time event stream in JSON format, to support analytics tools across a range of hardware interfaces.

2.6 Lessons Learned: Guiding Principles for Extensible Student Modeling

In designing *CT+A* so it can support an open range of student modeling applications, with provisions for real-time support tools, a set of guiding architectural principles has emerged. These principles capture the key architectural elements added to *CT+A.*

Maximize tutor-side computations. We have structured detectors to promote incremental (e.g., per transaction) computation of analytics. This supports offloading of student model computations to the tutor clients, rather than the LMS, since incremental computations spread processing load over time.

Keep data streams "lean". In designing key data streams (i.e., the transaction stream into the detectors, and the student model update stream from tutor to LMS), we settled on a small subset of the information *CTAT* tutors currently send to *LearnSphere* [24]. We originally attempted to anticipate many possible author needs and build these into the transaction messages [36] that serve as primitive inputs to plug-in detectors, but decided against this approach. Keeping this set small can reduce unnecessary message traffic and redundancy by acknowledging the wide range of analytics authors may wish to compute and enabling them to compute only those needed.

Maintain the student model both locally and centrally. Prior to these architectural extensions, an up-to-the-second copy of the student model was maintained on the tutor side, but the LMS-side copy was updated only as needed to preserve the student model in-between problems. We have found it valuable to instead maintain both a local (tutor-side) and central (LMS-side) up-to-the-second copy of the student model, with each copy supporting different use cases, namely, tutor adaptivity versus analytics tools; the latter typically require both class-level analytics and real-time updating, which is why central copies of the student models are useful.

Support easy re-mixing of existing components. In addition to supporting plug-and-play of architectural components, we have found it valuable to make individual components easily-customizable. For example, each detector contains a module that exposes configurable parameters. This feature is intended to facilitate the creation of *variants* of student modeling techniques, including those created and shared by others, to support authors not only in comparing against each other's' models, but also in *building upon and contributing to* each other's modeling work (c.f. [22, 37, 38]).

3 Case Studies

In this section, we present case studies of prototype systems that use the *CT+A* architecture to enhance tutoring systems' adaptive capabilities and/or to support teachers.

3.1 A Prototype Tutor that Provides Metacognitive Scaffolding

The experience of some of the participants during our yearly summer school illustrates how a detector library can be helpful in quickly prototyping adaptive tutor behaviors. During this summer school, designers, teachers, and researchers build their own systems using *CT+A*. This past year, participants were able to author detector-enhanced ITSs, by re-using pre-existing detectors available in the detector library. They embedded pre-existing detectors into their tutors and authored custom adaptive tutor behavior based on detectors' outputs.

A team of two students, Dennis Bouvier and Ray Martinez, used the *CT+A* architecture to implement an ITS prototype that provided metacognitive feedback in addition to feedback at the domain level, which is standard in *CTAT* tutors. This tutor, the Binary Search tutor, was intended to help undergraduate Computer Science students learn binary search algorithms. It allows students to practice applying a binary search algorithm to an array of numbers. The *Binary Search Tutor* uses a plug-in implementation of the Help Model, which can identify patterns in student-tutor interactions that

indicate *abuse* (e.g., rapidly clicking through hints without reading) or *avoidance* (e.g., not using hints in situations where they are likely to be needed) of the tutoring software's built-in hints [3]. Using custom response actions authored in the Tutor's Ear, the *Binary Search Tutor* responds to both types of student behavior. In the case of hint avoidance, the tutor prompts the student to ask for a hint. In the case of hint abuse, the tutor encourages the student to try attempting more steps without hints.

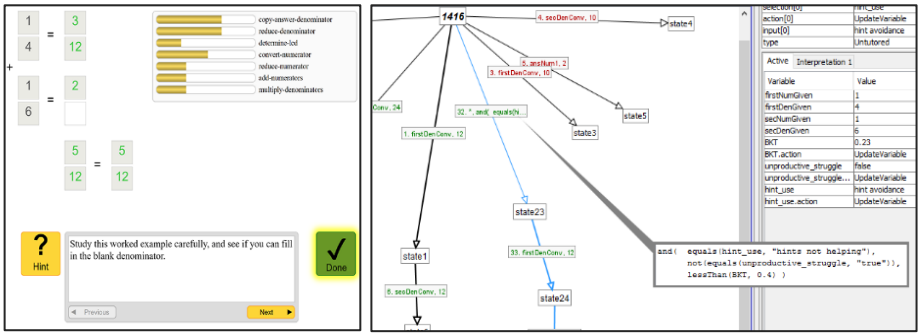

Fig. 2. Left: The *Fraction Addition Tutor* uses multiple plug-in detectors to decide whether to provide more scaffolding. Right: Authoring the *Fraction Addition Tutor* in *CTAT*.

3.2 A Prototype Fraction Addition Tutor with Hybrid Adaptivity

Using *CT+A*, we have also created a tutor prototype that implements a form of "hybrid adaptivity" [2], meaning that it adapts to combinations of student states. This tutor, an example-tracing tutor for 4th and 5th grade fraction addition problems, adjusts the level of scaffolding provided based *jointly* on the values of cognitive variables (skill mastery) and metacognitive variables (hint use, unproductive persistence). For example: if a student is detected as having low knowledge on KCs involved in the current step (by a plug-in of BKT [8]) *and* the student is detected as "using all available hints yet remaining stuck" (by the Help Model [3]) *but* the student is not currently detected as necessarily "unproductively persisting" (by a detector of wheel-spinning [21]), then the *Fraction Addition Tutor* will dynamically convert the student's current problem into a completion problem, by filling out all steps except one, and prompt the student to study the worked-out steps and fill in the remaining step (Figure 2, left). This capability was authored using a formula (expressed in *CTAT's* formula language) that references student model variables (i.e., the first of the two mechanisms described above for authoring adaptive tutor behavior). This formula was attached to a new path in the behavior graph (the main representation of domain knowledge in an example-tracing tutor), added by the author (Figure 2, right). The path specified the tutor-performed actions needed to fill in the worked-out steps.

3.3 Teacher Smart Glasses that Support Real-time Classroom Orchestration

The *CT+A* architecture has been used to implement *Lumilo* [17, 19], a mixed reality smart glasses application, co-designed with K-12 math teachers, and developed for

the Microsoft *HoloLens* [28]. *Lumilo* is designed to aid teachers in orchestrating personalized class sessions, in which students work with ITSs at their own pace. When a teacher puts these glasses on, she/he can see visual indicators floating over students' heads (Figure 3), based on changes in a student's extensible student model. The teacher can also view more detailed student-level analytics, as well as class-level summaries. *Lumilo* has been used in fifteen K-12 classrooms so far [17, 19].

All student model updates are computed within students' tutor clients (using several plug-in detectors) and forwarded to *TutorShop*, which forwards them to *Lumilo*. Although *Lumilo* is not browser-based (and was thus authored outside of *Live Dashboard*, described above), *TutorShop* provides hooks for *Lumilo* to connect to each classrooms' analytics streams. *Lumilo*'s dashlets are then updated by aggregators on the *Lumilo* client.

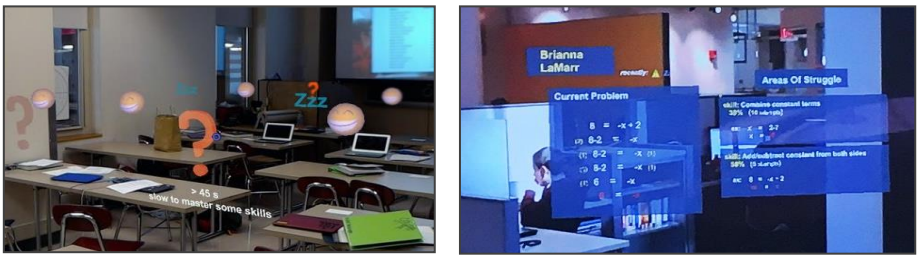

Fig. 3. Left: Point-of-view screenshot of teacher using *Lumilo* to monitor a class of students (taken directly after class). Right: Teacher's view through *Lumilo* after selecting a student, to view more detailed information for that student [17].

3.4 A Tablet-based Real-time Dashboard for Personalized Class Sessions

In addition to the smart glasses interface of *Lumilo*, a tablet-based companion app is being developed within the *Live Dashboard* and *AggHouse* tools. The tablet companion to *Lumilo* provides the same analytics and allows teachers to toggle between alternative display formats. For example, teachers can use *Live Dashboard*'s TableView component to display student model updates in a student-by-variable matrix format. Alternatively, teachers can use *Live Dashboard*'s Seating-Chart component to display a "real-time, real-place" visualization [18, 41] of the classroom, using a teacher-provided seating chart, and draggable student components (Figure 4).

3.5 A Prototype Dashboard that Supports Data-informed Lesson Planning

The *CT+A* architecture was used to develop *Luna*, a prototype browser-based dashboard front-end for K-12 teachers. Unlike *Lumilo*, which is designed to support real-time monitoring, Luna supports teachers in lesson planning, using analytics generated by an ITS for algebraic equation solving [20, 42]. *Luna* allows teachers to review students' knowledge and amount of practice on each of a number of fine-grained skills and error categories, either at the level of a class summary, or at the individual student level. In addition, teachers can use *Luna* to review individual students' progress through the software, relative to the time they have spent working (Figure 4).

Luna was developed using the *Live Dashboard* and *AggHouse* tools. As with *Lumilo*, the primitive level of data upon which *Luna* relies are student model updates, computed by plug-in detectors which are distributed across students' client machines.

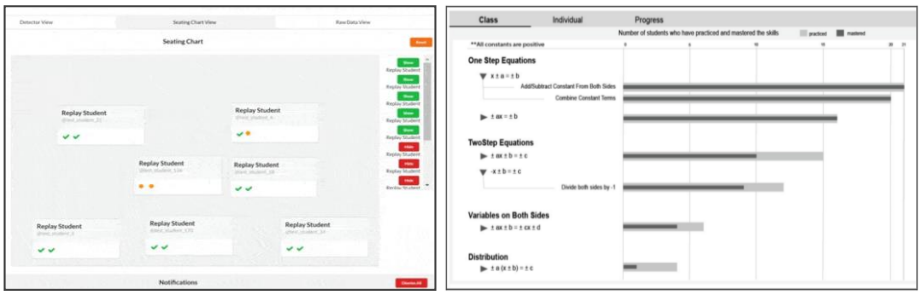

Fig. 4. Left: prototype of the *Lumilo Tablet* real-time dashboard, with students displayed as blocks within a *Live Dashboard* "Seating Chart" component. Right: a prototype of the *Luna* lesson-planning dashboard, showing the class-level view.

3.6 A Fractions Tutor with a Custom Adaptive Task Selection Policy

Finally, the *CT+A* architecture was used to develop an adaptive fractions tutor [12, 13] which can use a variety of custom instructional policies [12] to drive adaptive task selection (e.g., adaptive policies learned via reinforcement learning). The *Fractions Adaptive* tutor makes its student model available to external, custom task selection processes (Python web applications) via the *TutorShop* LMS. *TutorShop*, in turn, selects a next task for each student based on the output of this plug-in task loop.

4 Discussion and Future Work

If advances in student modeling made by the AIED, EDM, and LAK communities are to have a measurable impact on the design and effectiveness of real-world systems, and contribute to a *cumulative* science of student modeling, it is critical to develop authoring tools that can support these goals. Toward this end, we have introduced *CT+A*, an open architecture to support extensible student modeling. This architecture supports the plugging in, sharing, re-mixing, and use of advanced student modeling techniques in ITSs and associated analytics tools. The work is unique in that it supports extensible student models in the context of non-programmer ITS authoring tools that support building tutors with a dedicated problem-solving interface and elaborate step loop. In addition to the architecture itself, we present a set of "lessons learned," in the form of principles summarizing the main architectural elements. We hope they will inform other projects focused on extensible student modeling.

Our case studies illustrate some of the range and flexibility of *CT+A* and demonstrate progress towards four key goals for an analytics-integrated architecture. We have demonstrated that authors can add new variables to the student model by embedding detectors in running tutoring systems. We have presented an API and tem-

plate for creating these plug-in detectors, requiring only that authors are familiar with basic JavaScript. We demonstrated as well that existing detectors can be reused and that authoring new adaptive tutoring behavior is feasible without programming. Finally, we have shown that the *CT+A* architecture can support the development of a variety of teacher support tools, including both real-time and lesson-planning dashboards, and both web-based and wearable tools.

Limitations of the work are, at least for the time being, that we focus on transaction-based (in other words, sensor-free) student modeling [11]. Although transaction-based student modeling is a practical, proven, and widely useful approach (e.g., [4, 11, 15, 25, 38]), we leave for future work any issues related to how a student model can be updated with multiple data streams of different granularity (transactions and sensor output). As mentioned, such issues are being explored in the *GIFT* architecture [37]. An additional limitation of the current architecture is that, in authoring tutoring behaviors responsive to the extensible student model, immediate tutor responses involve a different mechanism than tutor responses in subsequent tutor cycles. A more flexible and general solution might be give detectors and the tutor engine equal status, with a coordinating agent that has the final word regarding the tutor response [36]. Finally, adding student model extensions requires some programming (namely, to create detectors in Javascript) and thus falls outside *CTAT*'s non-programmer paradigm. The amount of programming required can be reduced, however, by re-using existing detectors, shared among authors in the *CT+A* detector library [9]. In the future, new practices developed and tested within architecture might inform extensions to support their use without programming.

It is our hope that *CT+A* will help lower the barriers to sharing advanced student modeling methods between researchers, which in turn may accelerate progress within a cumulative science of student modeling (c.f., [11, 32, 37]). Support for plugging in – and sharing student modeling methods – can support authors and researchers not only in comparing against each other's' models (e.g., by evaluating systems that use these models in classroom experiments), but even in *building upon and contributing to* others' student modeling work (c.f., [22, 37, 38]). Also, they might help increase the number of close-the-loop studies that researchers undertake. We also hope that architectures like *CT+A* will result in broader representation of advanced student modeling methods in both research and real-world educational software.

Acknowledgements

This work was supported by NSF Award #1530726, and IES Grant R305B150008 to CMU. Opinions are those of the authors and do not represent views of NSF, IES or the U.S. ED. Special thanks to Ryan Baker, Nupur Chatterji, Mary Beth Kery, Michal Moskal, Luc Paquette, Peter Schaldenbrand, Cindy Tipper, and Franceska Xhakaj.

References

1. Aleven V., McLaren B.M., Sewall J., van Velsen M., Popescu, O., Demi, S., Ringenberg,

M. and Koedinger, K.R.: Example-Tracing tutors: Intelligent tutor development for non-programmers. *IJAIED*. 26, pp. 224-269 (2016).
2. Aleven, V., Mclaughlin, E.A., Glenn, R.A. and Koedinger, K.R.: Instruction based on adaptive learning technologies. *Handbook of research on learning and instruction*. Routledge (2017).
3. Aleven, V., Roll, I., McLaren, B.M. and Koedinger, K.R.: Help Helps, but only so Much: Research on Help Seeking with Intelligent Tutoring Systems. In: *IJAIED*. 26, 1, pp. 205–223. (2016).
4. Baker, R.D., Corbett, A., Koedinger, K.R., Evenson, S., Roll, I., Wagner, A.Z., Naim, M., Raspat, J., Baker, D.J. and Beck, J.E.: Adapting to when students game an intelligent tutoring system. In: *ITS*. pp. 392–401 (2006).
5. Blessing, S.B., Aleven, V., Gilbert, S., Heffernan, N.T., Matsuda, N. and Mitrovic, A.: Authoring example-based tutors for procedural tasks. In: *Design recommendations for adaptive intelligent tutoring systems*. US Army Research Laboratory. pp. 71–93 (2015).
6. Cen, H., Koedinger, K., & Junker, B.: Learning factors analysis – a general method for cognitive model evaluation and improvement. In: *ITS*. pp. 164-175. Springer, Berlin, Heidelberg (2006).
7. Clow, D. The learning analytics cycle. *LAK*. pp. 134–138. ACM (2012).
8. Corbett, A.T. and Anderson, J.R.: Knowledge tracing: Modeling the acquisition of procedural knowledge. *UMUAI*. 4, 4, pp. 253–278 (1995).
9. CT+A - Detector plugins, https://github.com/d19fe8/CTAT-detector-plugins/wiki/.
10. D'Mello, S.K., Lehman, B. and Graesser, A.: A motivationally supportive affect-sensitive AutoTutor. *New perspectives on affect and learning technologies*, pp. 113–126 (2011).
11. Desmarais, M.C. and Baker, R.S.J.D.: A review of recent advances in learner and skill modeling in intelligent learning environments. *UMUAI*. 22, pp. 9–38 (2012).
12. Doroudi, S., Aleven, V., & Brunskill, E.: Robust evaluation matrix: Towards a more principled offline exploration of instructional policies. In *L@S*. pp. 3-12. ACM (2017).
13. Doroudi, S., Holstein, K., Aleven, V., & Brunskill, E.: Sequence Matters, But How Exactly? A Method for Evaluating Activity Sequences from Data. In: *EDM*. pp. 70-77 (2016).
14. Fancsali, S.: Causal Discovery with Models : Behavior, Affect, and Learning in Cognitive Tutor Algebra. In: *EDM*. pp. 28–35 (2014).
15. Fancsali, S.E., Ritter, S., Stamper, J. and Nixon, T.: Toward "Hyper-Personalized" Cognitive Tutors: Non-Cognitive Personalization in the Generalized Intelligent Framework for Tutoring. In: *AIED*. pp. 71–79 (2013).
16. Grawemeyer, B., Holmes, W., Gutiérrez-Santos, S., Hansen, A., Loibl, K., & Mavrikis, M.: Light-bulb moment?: Towards adaptive presentation of feedback based on students' affective state. In: *IUI*. pp. 400-404. ACM (2015).
17. Holstein, K., Hong, G., Tegene, M., McLaren, B. M., & Aleven, V.: The classroom as a dashboard: Co-designing wearable cognitive augmentation for K-12 teachers. In: *LAK*. pp. 79-88. ACM (2018).
18. Holstein, K., McLaren, B.M. and Aleven, V.: SPACLE: Investigating learning across virtual and physical spaces using spatial replays. In: *LAK*. pp. 358–367 (2017).
19. Holstein, K., McLaren, B.M., & Aleven, V.: Student learning benefits of a mixed-reality teacher awareness tool in AI-enhanced classrooms. To appear: *AIED* (2018).
20. Holstein, K., Xhakaj, F., Aleven, V. and Mclaren, B.M.: Luna: A Dashboard for Teachers Using Intelligent Tutoring Systems. In: *IWTA@EC-TEL* (2016).
21. Kai, S., Almeda, V.A., Baker, R.S., Shechtman, N., Heffernan, C., Heffernan, N.: Modeling wheel-spinning and productive persistence in Skill Builders. To appear: *JEDM* (2018).

22. Kery, M.B. and Myers, B.A.: Exploring Exploratory Programming. In: *VL/HCC*. pp. 25-29 (2018).
23. Khajah, M., Lindsey, R. V and Mozer, M.C.: How Deep is Knowledge Tracing? In: *EDM*. pp. 94–101 (2015).
24. Koedinger, K.R., Baker, R.S.J.D., Cunningham, K., Skogsholm, A., Leber, B. and Stamper, J.: A Data Repository for the EDM community: The PSLC DataShop. In: *Handbook of Educational Data Mining*. CRC Press. pp. 43–56 (2010).
25. Liu, Z., Pataranutaporn, V., Ocumpaugh, J. and Baker, R.S.J.D.: Sequences of Frustration and Confusion, and Learning. In: *EDM*, pp. 114–120 (2013).
26. Long, Y. and Aleven, V.: Supporting Students' Self-Regulated Learning with an Open Learner Model in a Linear Equation Tutor. In: *AIED*. 219–228 (2013).
27. MacLellan, C. J., Koedinger, K. R., & Matsuda, N.: Authoring tutors with SimStudent: An evaluation of efficiency and model quality. In: *ITS*. pp. 551-560. Springer, Cham (2014).
28. Microsoft HoloLens, https://www.microsoft.com/en-us/hololens.
29. Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., & McGuigan, N.: ASPIRE: an authoring system and deployment environment for constraint-based tutors. In: *IJAIED*, 19, 2. 155-188 (2009).
30. Node.js, https://nodejs.org/en/.
31. Olsen, J., Belenky, D., Aleven, V., Rummel, N., Sewall, J., & Ringenberg, M.: Authoring tools for collaborative intelligent tutoring system environments. In: *ITS*. pp. 523-528. Berlin: Springer (2014).
32. Paquette, L., Baker, R.S., and Moskal, M.: A System-General Model for the Detection of Gaming the System Behavior in CTAT and LearnSphere. To appear: *AIED* (2018).
33. Pavlik, P. I., Cen, H., & Koedinger, K. R.: Performance factors analysis - A new alternative to knowledge tracing. In: pp. 531-538. Amsterdam: IOS Press (2009).
34. Pelánek, R. and Řihák, J.: Experimental Analysis of Mastery Learning Criteria. *UMAP*. pp. 156–163 (2017).
35. 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 tutoring system content creation. *IEEE TLT*, 2, 2. pp. 157-166 (2009).
36. Ritter, S. and Koedinger, K.: Towards Lightweight Tutoring Agents. In: *AIED*. pp. 16–19 (1995).
37. Sottilare, R.A., Baker, R.S., Graesser, A.C. and Lester, J.C.: Special Issue on the Generalized Intelligent Framework for Tutoring (GIFT): Creating a Stable and Flexible Platform for Innovations in AIED Research. *IJAIED*. pp. 1–13 (2017).
38. Stamper, J., Koedinger, K., Pavlik Jr, P. I., Rose, C., Liu, R., Eagle, M., & Veeramachaneni, K.: Educational Data Analysis using LearnSphere. In *EDM 2016 Workshops and Tutorials* (2016).
39. VanLehn, K.: The behavior of tutoring systems. *IJAIED*. 16, 3. pp. 227-265 (2006).
40. VanLehn, K.: Regulative loops, step loops and task loops. *IJAIED*. 26, 1, pp. 107-112 (2016).
41. Vatrapu, R.K., Kocherla, K. and Pantazos, K.: iKlassroom: Real-Time, Real-Place Teaching Analytics. *IWTA@LAK* (2013).
42. Xhakaj, F., Aleven, V. and McLaren, B.M.: Effects of a Teacher Dashboard for an Intelligent Tutoring System on Teacher Knowledge, Lesson Planning, Lessons and Student Learning. *EC-TEL*. pp. 315–329 (2017).
43. Yudelson, M. V., Koedinger, K.R. and Gordon, G.J. Individualized bayesian knowledge tracing models. *AIED*. pp. 171–180 (2013).