Using HCI Task Modeling Techniques to Measure How Deeply Students Model

Sylvie Girard, Lishan Zhang, Yoalli Hidalgo-Pontet, Kurt VanLehn, Winslow Burleson, Maria Elena Chavez-Echeagary, Javier Gonzalez-Sanchez

Arizona State University, Computing, Informatics, and Decision Systems Engineering, Tempe, AZ, 85281, U.S.A.

{sylvie.girard, lzhang90, yhidalgo, kurt.vanlehn, winslow.burleson, helenchavez, javiergs}@asu.edu

Abstract: User modeling in AIED has been extended in the past decades to include affective and motivational aspects of learner's interaction in intelligent tutoring systems. An issue in such systems is researchers' ability to understand and detect students' cognitive and meta-cognitive processes while they learn. In order to study those factors, various detectors have been created that classify episodes in log data as gaming, high/low effort on task, robust learning, etc. When simulating students' learning processes in an ITS, a question remains as to how to create those detectors, and how reliable their simulation of the user's learning processes can be. In this article, we present our method for creating a detector of shallow modeling practices within a meta-tutor instructional system. The detector was defined using HCI (human-computer interaction) task modeling as well as a coding scheme defined by human coders from past users' screen recordings of software use. The detector produced classifications of student behavior that were highly similar to classifications produced by human coders with a kappa of .925.

Keywords: intelligent tutoring system, shallow learning, robust learning, human-computer interaction, task modeling

1 Introduction

Advances in student modeling in the past two decades enabled the detection of various cognitive [3, 4, 8, 11, 13, 16, 17], meta-cognitive [1,6], and affective [2, 9] processes during learning based on classification of episodes in log data. Steps have been taken toward detecting when learning occurs [4] and to predict how much of the acquired knowledge students can apply to other situations [5, 6]. However, an obstacle in such research is how to gain an understanding of the user's cognitive or meta-cognitive processes while learning. While some of the indicators used in the literature are common to any intelligent tutoring system, others are closely linked to the activities and pedagogical goals of a specific application. The adaptation of such indicators to the design of a new system often necessitates a detailed analysis of the new domain and how the tutoring system guides learners to acquire its skills and knowledge. In particular, an issue within this process is the ability to reach common ground between learner scientists that perform an analysis of learners (meta-)cognitive actions at a high level - via video or log analysis of student's past actions for example – and the definition of the indicators by software engineers, related to how the system was implemented, that can be used to simulate such processes in agreement with the constraints and functionalities of software. We view the specificity of detectors as unavoidable, so the best solution is to develop good methods for analyzing the new tutoring system and designing the detectors. This short article describes our method and its application to out project, AMT. In the AMT project, a choice was made to use HCI (human computer interaction) task modeling - a method for formally representing human activity, and by extension, the behavior of an interactive system -, as well as video coding schemes from human coders, to develop the detectors. The detectors aim to evaluate student's use of shallow and deep modeling practices with and without being guided by a meta-tutor, on the domain of dynamic systems modeling.

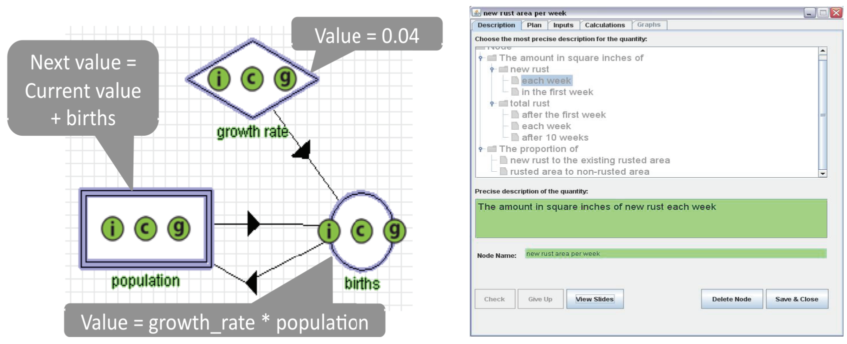
In Section 2, the AMT learning environment, for which the detectors were created, is introduced. In a third section, the task model of the user's activity in AMT is described. Next, the process of defining a coding scheme for the detector with human coders is presented, followed by the definition of the different classifications that define the value, the implementation and empirical evaluation of the detector. The final section summarizes the uses of task modeling within this work, and how it could be applied in future to other applications.

2 AMT software: a meta-tutor to teach deep modeling of dynamic systems.

AMT software teaches students how to create and test a model of a dynamic system. In our modeling language, a model is a directed graph with one type of link, as illustrated in Figure 1. Each node represents both a variable and the computation that determines the variable's value. There are three types of nodes.

- A *fixed value* node represents a constant value that is directly specified in the problem. A fixed value node has a diamond shape and never contains incoming links.
- An *accumulator* node accumulates the values of its inputs. That is, its current value is the sum of its previous value plus or minus its inputs. An accumulator node has a rectangular shape and always has at least one incoming link.
- A *function* node's value is an algebraic function of its inputs. A function node has a circular shape and at least one incoming link.

The students' learning objective is to draw a model representing a situation that is described in the form of a relatively short text. In the example of Figure 1, the description of the problem was " *Rust destroys steel and can spread quickly. Suppose you take a large sheet of steel, such as one that might be used as the roof of the boxcar on a train, and you put it outside in the weather. Suppose it starts with a spot of rust that is 10 square inches in area. However, each week the rust spot gets bigger, as it grows by 30%. Therefore at the end of the first week, the rust spot is 13 square inches in area.*" and the objective of the problem was to "Graph the size of the rust spot over 10 weeks."



Choose the most precise description for the quantity:

- The amount in square inches of
  - new rust
    - each week
    - in the first week
  - total rust
    - after the first week
    - each week
    - after 10 weeks
- The proportion of
  - new rust to the existing rusted area
  - rusted area to non-rusted area

Precise description of the quantity:

The amount in square inches of new rust each week

Node Name: new rust area per week

Fig. 1. The left image is the example of model, with gray callouts added to explain the contents of nodes. The right image is the example of a node editor.

The student constructs the model node by node, by filling in all information within each node in the form of four interactive tabs (description, plan, inputs, and calculations). During construction, students can use the *Check* button to evaluate the correctness of the current tab, or the *Solve it for me* button to ask the system to fill out the tab automatically.

The instruction is divided into three phases: (1) an introduction phase where students learn basic concepts of dynamic system model construction and how to use the interface; (2) a training phase where students are guided by a tutor and a meta-tutor to create several models; and (3) a transfer phase where all scaffolding is removed from soft-ware and students are free to model as they wish. The tutor gives feedback and corrections on domain mistakes.

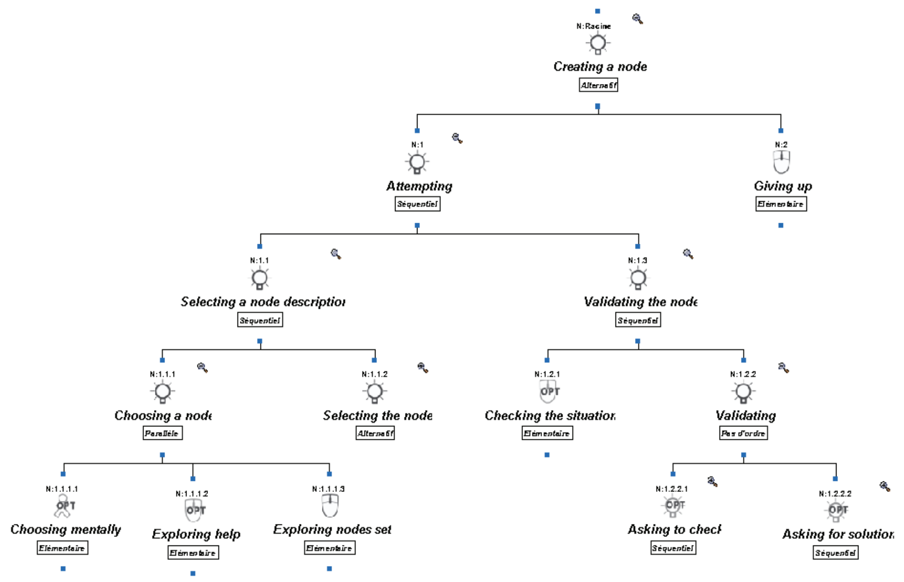
The meta-tutor requires students to follow a goal-reduction problem solving strategy, the Target Node Strategy [18]. The basic idea is to focus on one node at a time (the target node) and completely define it before working on any other node. This process decomposes the whole problem of modeling a system into a series of atomic modeling problems, one per node. Like *Pyrenees* [2], it teaches students that if they just master this one difficult but small skill, then the rest of the problem solving will be straight-forward. In addition, the meta-tutor complains if students appear to be guessing too much or giving up too early, just as the *Help Tutor* did [3].

While students learn, their motivation, attention to details, and modeling depth can fluctuate. To assess students, the project needed detectors that detect shallow and deep modeling practices both with and without the meta-tutor. The measure should be usable in the transfer phase of the experiment as a dependent variable, because deep modeling is the skill/knowledge that AMT teaches. The depth measure should also apply to student's behavior during the training phase so that we can check whether the instructional manipulations done during that phase have their intended effects (i.e., the measure serves as a manipulation check). The detector should further operate in real time (i.e., it doesn't require to know future actions or states in order to interpret the current action) so that it can be eventually be used by the system itself to condition its behavior.

3 Task Modeling: analysis of user's actions on software

A task model is a formal representation of the user's activity. It is represented by a hierarchical task tree to express all sub-activity that enables the user to perform the planned activity. The tasks need to be achieved in a specific order, defined in the task tree by the ordering operators. In AMT, every modeling activity follows the same procedure involving the same help features, task flow, and meta-tutor interventions. With a single task model of a prototypical modeling task, it is therefore possible to account for all of the user's activity in software. Due to the complexity of the final model, only one sub-activity will be described in this paper, illustrated in Figure 2. Only part of the model is deployed in the figure, and some subtasks will not be detailed here. In this part of the model the sub-activity the learner wishes to perform is to create a new node for the dynamic system s/he is currently modeling. We will first describe the task tree, and then insert the iterations and conditions that enable a formal verification of the flow of the task within the task model.

Figure 2: Sub-task "Creating a Node" in the AMT activity task model using K-MADe



Short description of the sub-task to model:

In order for a node to be created, the description tab of the node editor needs to be completed by selecting a node description, which corresponds to a valid quantity in the system to model. Each node is unique and cannot be created more than once. The user can engage in the task only if at least one node still needs to be created for the model to be complete.

Task tree and order of the tasks:

At the top level of the task tree "Creating a node", the learner can either attempt to create the node (task 1) or give up on the creation (task 2). The second task is represented in software by the user closing the node editor window, and can be done at any time during the task. The task "Creating a node" is over when a good description has been found and validated. The system can then try to initialize the selection and create the node.

In the first level of the task "Attempting", the learner first needs to select a node description (task 1.1), i.e.: what quantity the node will represent. S/he is then allowed to finish the creation of the node by validating the selection (task 1.2).

In order to select a node description, the user first needs to choose a node description (task 1.1.1) among the set of node descriptions offered by the system. This process involves the user choosing mentally one description (task 1.1.1.1), exploring the help features offered by software (task 1.1.1.2) and exploring the set of node descriptions displayed (task 1.1.1.3). S/he can then select the node (task 1.1.2). This subtask is not described in Figure 1 for a lack of space.

In order to validate the selection, the learner can choose to go back to the description of the problem to verify the correctness of his solution according to the problem to be simulated (task 1.2.1), and then has to validate the selection (task 1.2.1.2). When the user checks the validity of the selection, it can either be performed by checking the solution against the set of nodes still remaining to be modeled (task 1.2.1.2.1) or asking software to produce the solution (task 1.2.1.2.2). The user is allowed to ask for the solution only when a description has been checked at least once.

Now that the different actions of the learner are defined, the iterations and conditions will help represent the flow of the activity on the subtask "Selecting a node description" (task 1.1).

Iterative and Optional tasks

- Task 1.1 is iterative: it is possible to make several selections before trying to finish the description by validating.
- Task 1.1.1.2 is optional: The learner is not forced to explore the help features to choose a description, this is merely a choice on the learner's part.
- The main task, "creating a node", is iterative until the node is created or the activity is abandoned. The later is represented in the task model by an interruptible task: the learner can stop his/her creation of node activity any time by choosing to close the node editor window.

Conditions on tasks:

- Main task 1 has a pre-condition attached to it: the software only allows the user to engage in a creation of a new node if there is at least one node related to the modeling of the dynamic system that still remains to be created.

A first task model was created to represent learner's activity on software without the presence of the meta-tutor. This corresponds to the first version of software, which was evaluated against the interface including the meta-tutor in [18]. This second software interface includes a text-based agent that intervenes as the students engage in modeling to help them achieve deeper modeling behaviors, by applying constraints to the user's actions and giving meta-cognitive feedback. The meta-tutor was therefore added to the task model under the type "system" and the model was completed to include the constraints and interventions of the meta-tutor.

The final task model produced represented all possible actions of the learner on software in order to model a dynamic system. Next, a study of these actions, which led to the definition of the depth detectors, is detailed.

4 Detecting when students are modeling using shallow practices

The task model developed with K-MADe was used to define the episode structure. The first step in creating a coding scheme is to define a unit of measurement for the user's modeling actions. The task model clearly highlighted the different sub-activities the learner could engage in, referred to as goals. All goals are interruptible tasks in favor to accessing the help features[1] or abandoning the completion of the current goal for a new one. After a brainstorming session where researchers studied how students' actions fell in line with those goals, the following unit of depth, called "segment", was defined. This established the unit of coding to be used in the next phase.

Screen videos representing the learners' use of the AMT software with and without the meta-tutor were recorded during an experimental study described in [6]. These videos were studied to determine how much shallow vs. deep modeling occurred and the contexts, which tended to produce each type. A coding system was then created for video recordings of the learners' behavior. Three iterations of design for this coding scheme were performed, ending with a coding scheme that reached a multi-rater pairwise kappa of .902. The final coding scheme mapped learners' behavior to six classifications, which were implemented as the following depth detectors[AIED short paper]

- GOOD_METHOD: The students followed a deep method in their modeling. They used the help tools appropriately, including the one for planning each part of the model.
- VERIFY_INFO: Before checking their step for correctness, students looked back at the problem description, the information provided by the instruction slides, or the meta-tutor agent.

[1] It is to be noted that two help systems are available to users: (1) referring back to the instructions always available for viewing, and (2) looking at the problem situation where all details of the dynamic system to model are described.

- SINGLE_ANSWER: The student's initial response for this step was correct, and the student did not change it.
- SEVERAL_ANSWERS: The student made more than one attempt at completing the step. This includes guessing and gaming the system:
  - The user guessed the answer, either by clicking on the correct answer by mistake or luck, or by entering a loop of click and guessing to find the answer.
  - The user "games the system" by using the immediate feedback given to guess the answer: series of checks on wrong answers that help deduce the right answer.
- UNDO_GOOD_WORK: This action suggests a modeling misconception on the students' part. One example is when students try to run the model when not all of the nodes are fully defined.
- GIVEUP: The student gave up on finding how to do a step and clicked on the "give up" button.

Another detector was defined as a linear function of the six episode detectors. It was intended to measure the overall depth of the students' modeling, therefore providing an outcome measure in the transfer phase in future experimental studies. It considered two measures (GOOD_ANSWER, VERIFY_INFO) to indicate deep modeling, one measure (SINGLE_ANSWER) to be neutral, and three measures (SEVERAL_ANSWERS, UNDO_GOOD_WORK, and GIVE_UP) to indicate shallow modeling.

Once the coding scheme reached a sufficient level of agreement between coders, the task model was used to adapt the coding to students' actions on the software. The episodes that were coded for depth by human analysts in the sample video were analyzed by creating scenarios from the task model within K-MADe. The validation of six detectors' implementation involved three human coders, who watched a sample of 50 episodes, paying attention to the depth of modeling exhibited by the student's actions, and chose the classification that best represented the depth of the learner modeling at the time of the detected value. A multi-rater and pairwise kappa was then performed, reaching a level of inter-reliance of .925.

5 The different uses of the Task Model

The task modeling language K-MAD and its task model creation and simulation environment, K-MADe [7] were chosen for the following reasons: the environment enables the creation and replay of scenarios of student's actions, a set of functionalities not described here enable a formal verification of the model. Additionally the associated simulation environment ProtoTask [14] allows non-specialists in task modeling to visualize the flow of the task model, via scenarios in a clear and simple manner.

The use of K-MAD helped in the creation of the detectors and are a first step in offering an alternative technique to simulated learners, by tackling the following problems:

- *Breaching the gap between learner scientists' understanding of how the learning process works and programmers' definition of the application flow, functionalities, and indicators.*
- *Enabling a formal validation of software flow, understandable by all.*
- *Using simulated learners scenarios to define the detectors.*

A researcher in educational technology - expert in teaching modeling and part of the AMT project - and an HCI practitioner, realized the task model. The former was an expert on how AMT software was designed in terms of pedagogical content and task flow. His expertise focused in particular on the actions the students were allowed/incited/forbidden to do within software at each moment of the modeling task. The HCI practitioner was not familiar with intelligent tutoring systems or meta-tutors. She was involved in the creation of the task model in a consulting capacity, in regards to her expertise in task modeling of interactive systems.

The task model could be defined at the level of the user's planning of actions and system flow, with iterations and conditions alone. However, the objects in K-MADe enable us to represent the constraints of the learner's actions concretely and to apply a formal verification of task flow. It was therefore possible to represent the set of descriptions as either valid or invalid, to detect when a node has been checked and the result of that check, and to add constraints on the checking procedure such as to avoid node duplication. This enabled a formal verification of software flow prior to validate its fidelity to learner scientists' ideas about possible actions on software and the underlying processes involved.

Once the model was constructed, the use of ProtoTask to visualize software flow and follow learners' possible sets of actions allowed by software enabled the ability to simulate learners by creating scenarios of use that could be played and replayed at will, focusing on the cognitive and meta-cognitive levels of learner's experience on software. In the process of creating our detectors, a video analysis of learner's past actions was performed. The model could be used to check the possible actions of users with what the designer of the system wanted to offer as functionalities and software flow. During this analysis, the task model could be used once again to define scenarios that simulated learner's pertinent behaviors using ProtoTask. Once those scenarios were formed, the task analyst came back to the original K-MAD modeling language and studied the similarities and contrasts between scenarios to define the rules that govern the detection of shallow and deep modeling practices within AMT. Once the task model identified points of detection of such practices, it became easy for programmers to go back to software and implement the rules.

6 Conclusion and Future Work

In this paper, a method to create a detector of deep modeling within a meta-tutor using HCI task modeling and video coding schemes was described. The main outcome of this process was the creation of detectors inferring the depth of students' modeling practices while they learn on a meta-tutoring system, reaching a multi-rater and pairwise kappa score of .925. We believe the use of the task model to define shallow and deep modeling practices by helping to create the detectors to be of value for any simulated learning environments, in particular for indicators that a common to all learning tasks present in a tutoring system.

In interdisciplinary teams, the design of indicators can lead to communication issues due to misunderstandings and a lack of common ground between analysis made at a high level of learners' cognitive and meta-cognitive processes, and the representation of those behaviors within software. In particular, video-coding processes can become costly when the coders' understanding of the details of how the system works differs from how the system actually works. Our experience using K-MADe and ProtoTask highlighted an ease in this project in gaining a better view of the tutoring system and the detection of deep modeling within the interface. In particular, the use of ProtoTask by the non-specialists in task modeling helped clarify issues of task flow and the definition of the set of user's actions at each moment of interaction.

A limitation of the method is the applicability to different types of tutoring systems. In AMT, a single task model was able to represent the entirety of a users' learning activity. In tutoring systems that teach a set of skills through different pedagogical approaches for diverse types of learning tasks, the creation of such task models might prove more costly and may not be completely adapted to the creation of detectors that need to be adapted to each task specifically.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0910221. We would like to thank Sybille Caffiau for consulting in the project and sharing her expertise in task modeling of interactive systems.

References

1. Aleven, V., McLaren, B.M., Roll, I., Koedinger, K.R.(2006): Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor. International Journal of Artificial Intelligence and Education 16, 101–128
2. Arroyo, I., and Woolf, B.P., 2005. Inferring learning and attitudes from a Bayesian Network of log file data. In Proceedings of the 2005 conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology, Chee-Kit Looi, Gord McCalla, Bert Bredeweg, and Joost Breuker (Eds.). IOS Press, Amsterdam, The Netherlands, The Netherlands, 33-40.
3. Baker, R. S. J. d., Corbett, A. T., Koedinger, K. R., Evenson, S., Roll, I., Wagner, A. Z., … Beck, J. E. (2006). Adapting to when students game an intelligent tutoring system, Proceedings of the 8th international conference on Intelligent Tutoring Systems, Jhongli, Taiwan Berlin, Heidelberg.
4. Baker, R.S.J.d., Goldstein, A.B., Heffernan, N.T.: Detecting the Moment of Learning. In: Aleven, V., Kay, J., Mostow, J. (eds.) ITS 2010. LNCS, vol. 6094, pp. 25–34. Springer, Heidelberg (2010)
5. Baker, R. S. J. D., Gowda, S. M., & Corbett, A. T. (2011). Towards predicting future transfer of learning, Proceedings of the 15th international conference on Artificial intelli-

gence in education. Proceedings from AIED’11, Auckland, New Zealand Berlin, Heidelberg.

6. Baker, R. S. J. D., Gowda, S. M., Corbett, A. T., & Ocumpaugh, J. (2012). Towards automatically detecting whether student learning is shallow., Proceedings of the 11th international conference on Intelligent Tutoring Systems, Chania, Crete, Greece Berlin, Heidelberg.
7. Caffiau, S., Scapin, D., Girard, P., Baron, M., & Jambon, F. (2010). Increasing the expressive power of task analysis: Systematic comparison and empirical assessment of tool-supported task models. Interacting with Computers, 22(6), 569–593. doi:10.1016/j.intcom.2010.06.003
8. Corbett, A.T., MacLaren, B., Kauffman, L., Wagner, A., Jones, E.A.: Cognitive Tutor for Genetics Problem Solving: Learning Gains and Student Modeling. Journal of Educational Computing Research 42(2), 219–239 (2010)
9. D’Mello, S. K., Lehman, B., & Person, N. (2010). Monitoring affect states during effortful problem solving activities. International Journal of Artificial Intelligence in Education, 20(4), 361–389., doi:10.3233/JAI-2010-012
10. Girard, S., Chavez-Echeagary, H., Gonzalez-Sanchez, J., Hildalgo-Pontet, Y., Zhang, L., Burleson, W., and VanLehn, K., (2013), Defining the behavior of an affective learning companion in the affective meta-tutor project, in K. Yacef et al. (Eds.): Proceedings of the 16th international conference on Artificial Intelligence in EDucation (AIED’13), LNAI 7926, pp. 21--30. Springer-Verlag, Berlin, Heidelberg.
11. Gowda, S.M., Pardos, Z.A., and Baker, R. S. J. D. 2012. Content learning analysis using the moment-by-moment learning detector. In Proceedings of the 11th international conference on Intelligent Tutoring Systems (ITS'12), Stefano A. Cerri, William J. Clancey, Giorgos Papadourakis, and Kitty Panourgia (Eds.). Springer-Verlag, Berlin, Heidelberg, 434-443. DOI=10.1007/978-3-642-30950-2_56
12. Koedinger, K.R., Corbett, A.T., Perfetti, C. (2010): The Knowledge-Learning-Instruction (KLI) Framework: Toward Bridging the Science-Practice Chasm to Enhance Robust Student Learning. Carnegie Mellon University Technical Report, June, 2010
13. Martin, J., VanLehn, K.: Student assessment using Bayesian nets. International Journal of Human-Computer Studies 42, 575–591 (1995)
14. Lachaume, T., Girard, P., Guittet, L., & Fousse, A. (2012). ProtoTask, new task model simulator. In M. Winckler, P. Forbrig, & R. Bernhaupt (Eds.), Human-Centered Software Engineering (Vol. 7623, pp. 323– 330). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-34347-6
15. Muldner, K., Burleson, W., Van, D. S., Brett, & Vanlehn, K. (2011). An analysis of students’ gaming behaviors in an intelligent tutoring system: predictors and impacts. User Modeling and User-Adapted Interaction, April 2011, 21(1-2), 99–135m doi:10.1007/s11257-010-9086-0
16. Shih, B., Koedinger, K.R., Scheines, R.: A response time model for bottom-out hints as worked examples. In: Proceedings of the 1st International Conference on Educational Data Mining, pp. 117–126 (2008)
17. Walonoski, J. A., & Heffernan, N. T. (2006). Prevention of off-task gaming behavior in intelligent tutoring systems Proceedings of the 8th international conference on Intelligent Tutoring Systems. Jhongli, Taiwan Berlin, Heidelberg.
18. Zhang, L., Burleson, W., Chavez-Echeagary, H., Girard, S., Gonzalez-Sanchez, J., Hildalgo-Pontet, Y., and VanLehn, K., (2013), Evaluation of a meta-tutor for constructing models of dynamic systems, in K. Yacef et al. (Eds.): AIED’13, LNAI 7926, pp. 666--669. Springer-Verlag, Berlin, Heidelberg.