**Abstract:**

This document presents my understanding of the Pedagogical Model. I have given it some thought and have worked on the structure as well as modules that would be present in it.

**Terminology used:**

* Dragoon Java version is referred as Dragoon
* Dragoon JavaScript version is referred as Dragoon2 (to keep it in line with the original documentation)
* Student and User are used interchangeably.

**Pedagogical Model (PM)**

Pedagogical Model is the thinking module of the software (Dragoon 2). For a quick understanding, one can visualize the PM like Meta Tutor of Dragoon. It would consist of two primary parts –

1. Interpreter – for understanding the actions of the user
2. Decoder – this will use the Student profile and the interpretation to give response back to the front end to present it to the user.

Interpreter: Interacts with the front end and maintains the understanding of the actions taken by the user.

Decoder: Calculates the detector for the student implementation, understands the actions taken and maintains logs for the same.

Front-End: Interacts with the user.

Sends actions of the User

Receives actions to show to the user.

Pedagogical Model Interacting with the front end of the system

Interpreter:

This will continuously interact with the Front end and keep track of the user state. Handling the counters for showing the feedback to the user. It will connect the user actions and understand the bigger picture for creating a model. Like the nodes created and what initial values have been created. This will also handle the changes to the student profile and will update it as and when required for the counters and the problems student has already worked on.

Actions performed the interpreter:

1) Understanding the values selected by the user from the dropdown while creating/editing a node.

2) How the user has connected the nodes with other nodes.

3) Understanding the hidden function nodes. This will be true only for the power user though only after we have implemented that.

4) Whether the actions are in accordance with the test node strategy, and the correct nodes are being created.

5) What sort of hints are given to the user, and why were they needed. For eg. the previous action was wrong which led to creation of a wrong node or the student is not actually moving and taking a lot of time to think. All these actions will lead to providing a hint. Since there is no actual button to ask for a hint it will be presented on its own. It will interpret whether actions are right or wrong and then send it to the Decoder to decide based on the modes and the counters in the student profile to show the Hint or not.

Decoder:

This will get the step by step updates from the interpreter. It will compare it with the example model created during the author mode, and decide whether the student actions are right or wrong. It will be responsible for deciding the understanding of the student. Whether the student is following shallow modelling practices, or whether the student is diligently following the target node strategy.

**Gaming of the System:**

When a User wants to create a Node they use “Create Node” button present on the main screen. This opens up the Node Editor which had a Check button. There were some scenarios for gaming of the system like the user would check for the solution again and again very quickly without giving much thought and hence would reach the correct the solution. This kind of modelling is called shallow modeling as the user was not giving much thought. To handle such cases Finite State Machines as well as production rules were used. Both the scenarios with a wider idea of implementation will be presented below.



This is an example of Finite State Machine which was picked from Evaluation of Meta Tutor document.

Here we see if the check button is pressed continuously within three seconds and it’s a wrong check then meta tutor puts the user in S4 and it decides whether to send a message to the tutor via message channel or not. The messages are usually on the behavior of the user and hence the name – Meta Tutor.

In Dragoon 2 version the System does not have a check button. When the user opens the create node tab, and starts filling the details for creating the node, the pedagogical model checks the system on its own and comes back with the Red, Green, Blue or Yellow color for the hint. The user can still game the system in many ways. Although the complete hints will be just 2 in numbers, but the user can still try all the options (that are present in the dropdowns, chose them and wait for the color to turn green or red) and get away with shallow modeling practices. To check for such scenarios and come back with messages to explicate the behavior of the student, we can implement Finite State Machines or Production Rules.

Implementation of Finite State Machines:

We can use the original Finite State Machine, with some minor changes to handle the shallow modeling or gaming the system scenarios. The implementation in javascript we need to create objects for the states which have event listeners based on the divs. For the time taken we would need the relative time which can be picked up from the client and thus we just need to verify that whether it is greater than 3 seconds.

Implementation of Production Rules:

Production Rules are based on Facts. The facts here will be exact node changes, whether its correct or not, time taken between the previous step and the new step. We need some if and then rules to decide whether we need to show a message or not. These would be part of the client side implementation, which would be based on the events similar to the ones needed to create Finite State Machines.

Pros and Cons:

a) Implementation of Finite State Machine will need object creation for States and the precise definition. But this is a closer implementation of the real scenario.

b) Production Rules are easier to implement as compared to FSM as there are no objects required. All we need are the facts and the decision we would like to take based on them.

(I have written some of the rules and the new FSM in my copy. After verification I can put them in the document.)

**Measuring Student Modeling Techniques:**

As per the previous model used in Dragoon, we can continue to follow the same model. The definition for detectors used in it is presented below, with the changes for no tab Javascript implementation.

1) Good\_Method: The student takes his time to understand the problem. He creates nodes in the proper order (which will be how the author created the nodes while modelling the problem). The help tools were used, with proper time difference. There was no over usage of the calculate graph button or other check methods for the problem.

2) Verify\_Info: This would depend on the time student took to create the model and also how many incorrect options did the user try. If the incorrect options are not very quickly selected and the user isn’t just vaguely guessing then he will be placed under this detector.

3) Single\_Answer: This would mean that whatever steps student followed and the nodes that were created were in proper order. No help required and the user completed the problem very quickly.

4) Several\_Answers: This would mean that student did not try to understand the problem and went on guessing. He reached the solution with a number of clicks which is the primary case of Gaming the System.

5) Undo\_Good\_Work: This would mean that there is a modelling misconception with the student and the student is trying to run an incomplete model or he was correct initially but created extra nodes after that were not a part of the final solution.

6) GiveUp: This would be the case if the user gives up on the problem.

Here the deep modelling detectors will be Good\_Method and Verifiy\_Info whereas Several\_Answers, Undo\_Good\_Work and GiveUp will mean shallow modelling technique was followed. The Single\_Answer would mean the neutral modelling technique.

Based on the inputs from the interpreter, decoder module will decide the detector to define the student. This will be presented in the logs and would be part of the student profile.

The parameters required to distinguish the user in these detectors will be –

a) Time taken between each step

b) How many wrong or correct answers did the user chose

c) How many times the nodes were edited and how many total nodes were created.

Now these inputs will be implemented like normal production rules or we will implement Hidden Markov Model for calculating the detector in which the student lies is still under discussion.

**Database and Logging Structure for Dragoon 2:**

The database table structure is adopted from the table structure used in ANDES. The table structure was explained by Brett. I worked on some of the problems in ANDES and checked the logs as well.

The structure for Pedagogical Model was also picked from my understanding of ANDES system. ANDES had an interpreter which used to understand the actions taken by the user. The interpretation was then sent to the meta-tutor for comparison and provide a response to the user as per the interpretation of the actions. This helped in keeping the understanding and response modularized which could have changed with time. For actual functions of the interpreter module in PM please check the Page 2.

Database Table Structure

Table 1:

* tID – this is the serial number, which is the primary key for the table.
* SessionID – foreign key. This together with tID can also be used as a primary key even if tID is unique.
* method – this will be the action taken by the user and what has been interpreted from those actions. Here the actions wont be so small as how the mouse was moved but rather the bigger action. Like node was created with so and so values and whether it was in line with Target Node Strategy.
* Time – this will be the relative time since the student started working on the problem.

Table 2: this will be the session table of the user

* SessionID – primary key for this table
* Mode – user mode -> author, power, student or delayed feedback mode.
* Time – this will be the server time. So we will track the problem using the time in Table 1 which will start as the problem starts and the absolute time of the session will be saved here.
* User – the name and profile of the user.
* Class – section where the tutor is being used like whether it is CPI360 or sustainability.
* Problem – Problem name on which the user is working.

The table created in this format will actually be a JSON key-value at the end. These values will be added to another table which will just have the time, action and response as shown below. The response will have the json values which will explain the complete flow of the problem. It will have the action taken by the user as well as the interpretation done by the interpreter part of PM.

Now the logs for the software will be written by all the action interpretations of the user with the relative time from the start of the problem. Everything that will be entered in the database will be written like a JSON key-value pair. For eg:

|  |  |  |
| --- | --- | --- |
| **Time** | **Action** | **Response** |
| 0.001 | {"user":"sachin", "problem":"kr1b", "section":"andestutor.org", "extra":""} | * {"action":"new-object", "type":"statement", "id":"statement0", "mode":"locked", "x":10,"y":10,"width":400,"text":"A wheel is rotating clockwise at a constant angular velocity"} * {"action":"new-object", "type":"statement", "id":"statement1", "mode":"locked", "x":10,"y":35,"width":400,"text":"of 3\*π rad/s. What is the magnitude of the angular"} * {"action":"new-object", "type":"statement", "id":"statement2", "mode":"locked", "x":10,"y":60,"width":400,"text":"displacement of the wheel after 45.0 seconds?"} * {"action":"new-object", "type":"statement", "id":"statement3", "mode":"unknown", "x":40,"y":85,"width":100,"text":"Answer: "} * {"action":"new-object", "id":"graphic", "type":"graphics", "mode":"locked", "x":10,"y":110,"width":150,"height":142,"href":"../images/kr1b.gif"} * {"action":"new-object", "id":"time1", "type":"statement", "mode":"locked", "width":250,"x":450,"y":15,"text":"Time T0: start of problem."} * {"action":"new-object", "id":"time2", "type":"statement", "mode":"locked", "width":250,"x":450,"y":40,"text":"Time T1: 45 seconds later."} * {"action":"set-preference", "name":"angleSnap", "value":1} * {"action":"set-preference", "name":"informed-consent", "value":"agree:default-consent-asu"} * {"action":"set-score", "score":0} * {"action":"log", "log":"user-agent", "text":"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0"} * {"y":65,"x":450,"width":300,"text":"Let t01 be the duration of time between T0 and T1", "mode":"unknown", "id":"pre3", "action":"new-object", "type":"statement", "symbol":"t01"} * {"action":"modify-object", "id":"pre3", "mode":"correct"} |

This is how the problem started in ANDES. Here the JSON in the response holds the values that will be added to the Database.

Implementation:

1) It involves creating a table with Time, Action and Response columns. Time will be the relative time since the user has started the problem. This will help in the analysis of the problem as time taken by the user will define the understanding of Test Node Strategy of the user.

2) All the actions will documented using a json string which will have the key value pair as per the column name defined above. The values can be added as to another table as well as in the form of the json values as shown in the table. This will help in the analysis of the problem while finding average time of the class etc. ­­­­­­­­