Chatbot Architecture Document for MOOC Program

Objectives: Our Objective is to create and have a robust, interactive and dynamic Chatbot across all the MOOC Courses on EDX Platform. These chatbots should be able to:

- Evolve over time to meet student needs
- Evolve capabilities to improve performance and service
- Answer Program, Course and Certificate type questions
- Maintains its learning through training data (aka don't blow away the algorithm)
- Supportable by more than one developer

Architectural Elements of the Chatbot:

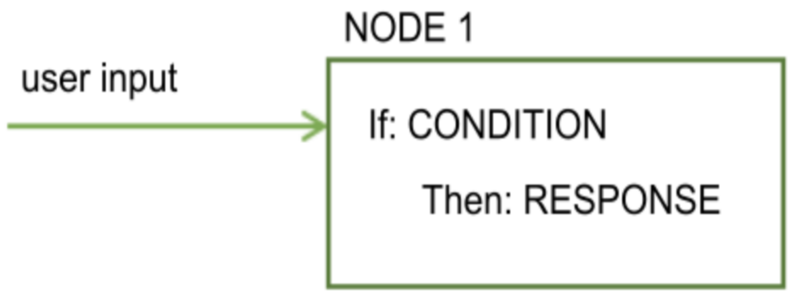
- **Intents:**
  - Intents are purposes or goals that are expressed in a customer's input, such as answering a question or processing a bill payment. By recognizing the intent expressed in a customer's input, the Watson Assistant service can choose the correct dialog flow for responding to it.
  - For eg. Let's say there is a bot operating for a library and you ask a question "Till what time is the library open"?. Here the intent of a customer is to know till what time is the library open so that he can come and either stay and read a book or issue one out. So we create an intent called "Open_time" where we can put all questions that a customer might ask related to open timings of the library.

- **Entities:**
  - Entities represent information in the user input that is relevant to the user's purpose.
  - If intents represent verbs (the action a user wants to do), entities represent nouns (the object of, or the context for, that action). For example, when the intent is to get a weather forecast, the relevant location and date entities are required before the application can return an accurate forecast.
  - So if you ask a question "What is the weather in college park"?. We can define an intent called "weather" where we will have all questions grouped together related to weather and temperature while we can have an entity defined @city that can be used to retrieve temperature of a place that a customer is interested in.
  - In this case it was college park, but in theory it can be anything like Dallas, NY, DC etc.

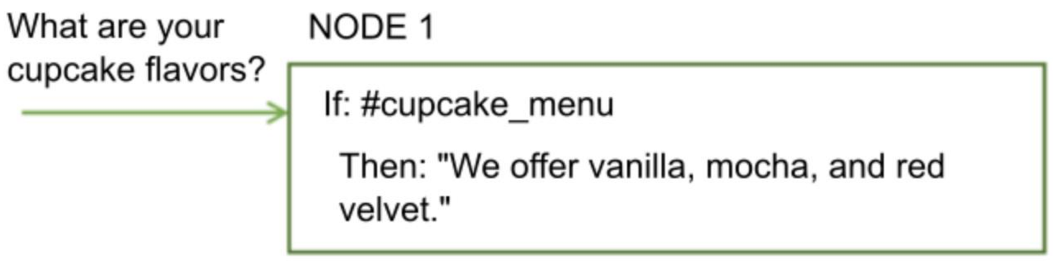
- **Dialogs:**
  - The dialog uses the intents that are identified in the user's input, plus context from the application, to interact with the user and ultimately provide a useful response.
  - The dialog matches intents (what users say) to responses (what the bot says back)
  - The response might be the answer to a question such as "Where can i get some gas?" as or the execution of a command, such as turning on the radio. The intent and entity might be enough information to identify the correct response, or the dialog might ask the user for more input that is needed to respond correctly

    - For example, if a user asks “Where can I get some food?” you might want to clarify whether they want a restaurant or a grocery store, to dine in or take out, and so on. You can ask for more details in a text response and create one or more child nodes to process the new input.

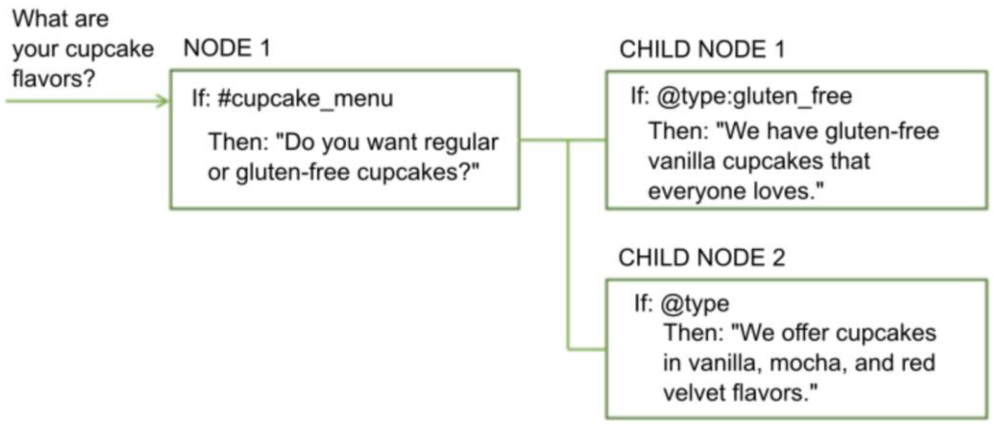
- **Dialog Nodes:**
    - Each dialog node has at minimum a condition and a response



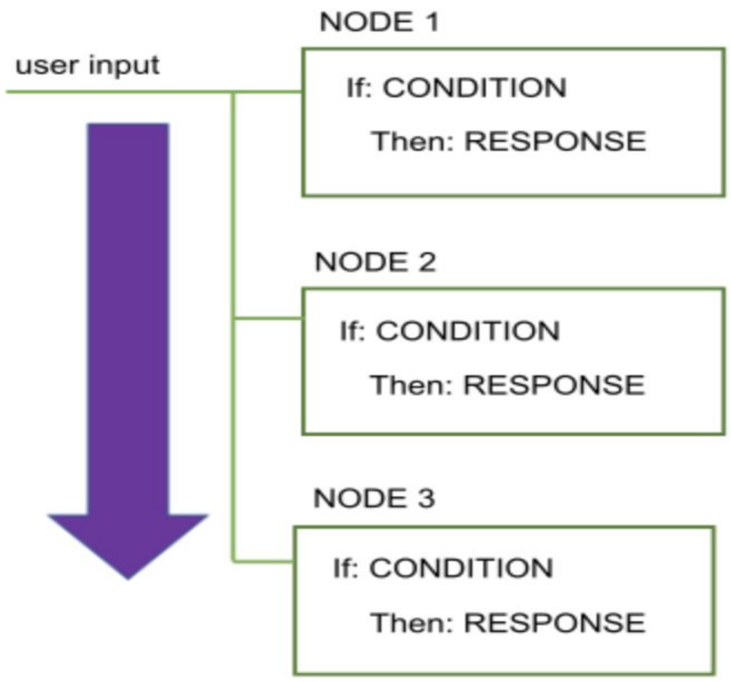
    - Condition: Specifies the information that must be present in the user input for this node in the dialog to be triggered. The information is typically a specific intent. It might also be an entity type, an entity value, or a context variable value.
    - Response: The utterance that your assistant uses to respond to the user. The response can also be configured to show an image or a list of options, or to trigger programmatic actions.You can think of the node as having an if/then construction: if this condition is true, then return this response.
    - For example, the following node is triggered if the natural language processing function of your assistant determines that the user input contains the #cupcake-menu intent. As a result of the node being triggered, your assistant responds with an appropriate answer.


What are your cupcake flavors?
NODE 1
If: #cupcake_menu
Then: "We offer vanilla, mocha, and red velvet."

    - A single node with one condition and response can handle simple user requests. But, more often than not, users have more sophisticated questions or want help with more complex tasks. You can add child nodes that ask the user to provide any additional information that your assistant needs.

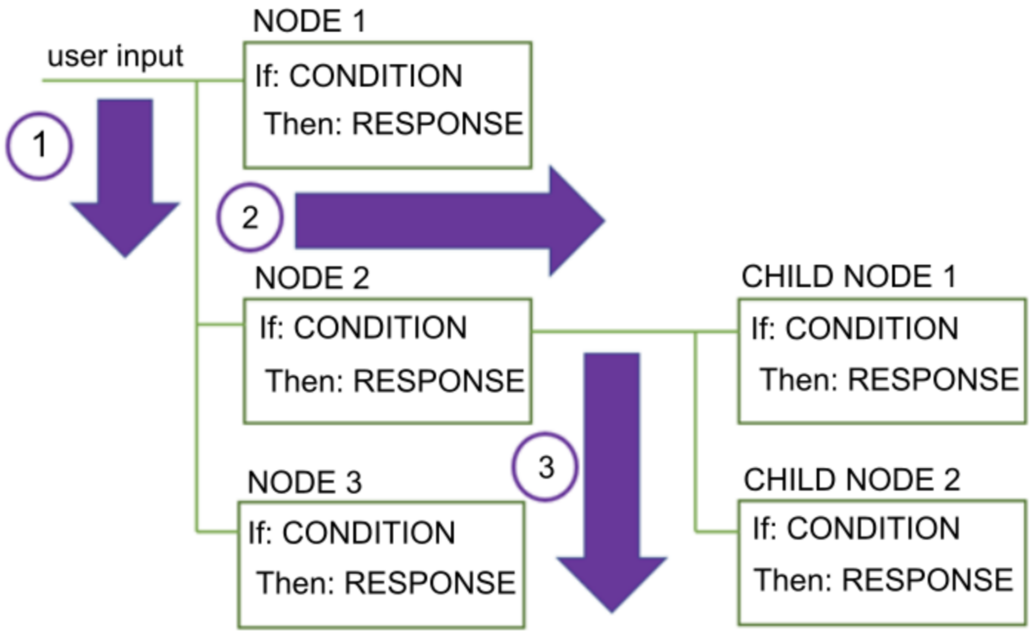

What are your cupcake flavors?
NODE 1
If: #cupcake_menu
Then: "Do you want regular or gluten-free cupcakes?"
CHILD NODE 1
If: @type:gluten_free
Then: "We have gluten-free vanilla cupcakes that everyone loves."
CHILD NODE 2
If: @type
Then: "We offer cupcakes in vanilla, mocha, and red velvet flavors."

- **Dialog Flow:**
  - The dialog that you create is processed by your assistant from the first node in the tree to the last.



  - As it travels down the tree, if your assistant finds a condition that is met, it triggers that node. It then moves along the triggered node to check the user input against any child node conditions. As it checks the child nodes it moves again from the first child node to the last.
  - Your assistant continues to work its way through the dialog tree from first to last node, along each triggered node, then from first to last child node, and along each triggered child node until it reaches the

last node in the branch it is following.



- When you start to build the dialog, you must determine the branches to include, and where to place them. The order of the branches is important because nodes are evaluated from first to last. The first root node whose condition matches the input is used; any nodes that come later in the tree are not triggered.
- When your assistant reaches the end of a branch, or cannot find a condition that evaluates to true from the current set of child nodes it is evaluating, it jumps back out to the base of the tree. And once again, your assistant processes the root nodes from first to the last. If none of the conditions evaluates to true, then the response from the last node in the tree, which typically has a special anything_else condition that always evaluates to true, is returned.
- You can disrupt the standard first-to-last flow in the following ways:
  - By customizing what happens after a node is processed. For example, you can configure a node to jump directly to another node after it is processed, even if the other node is positioned earlier in the tree.
  - By configuring conditional responses to jump to other nodes.
  - By configuring digression settings for dialog nodes. Digressions can also impact how users move through the nodes at run time. If you enable digressions away from most nodes and configure returns, users can jump from one node to another and back again more easily.

- **Naming**
  A bad naming convention can make a project look complex and increase development time. Hence, it is essential that we spend time on naming our intents, entities and dialog in a descriptive way that is easily understandable.

  - **Intents:** We must try to name our intent in such a way that it is:
    - Self-Descriptive
    - Unambiguous
    - Not repetitive

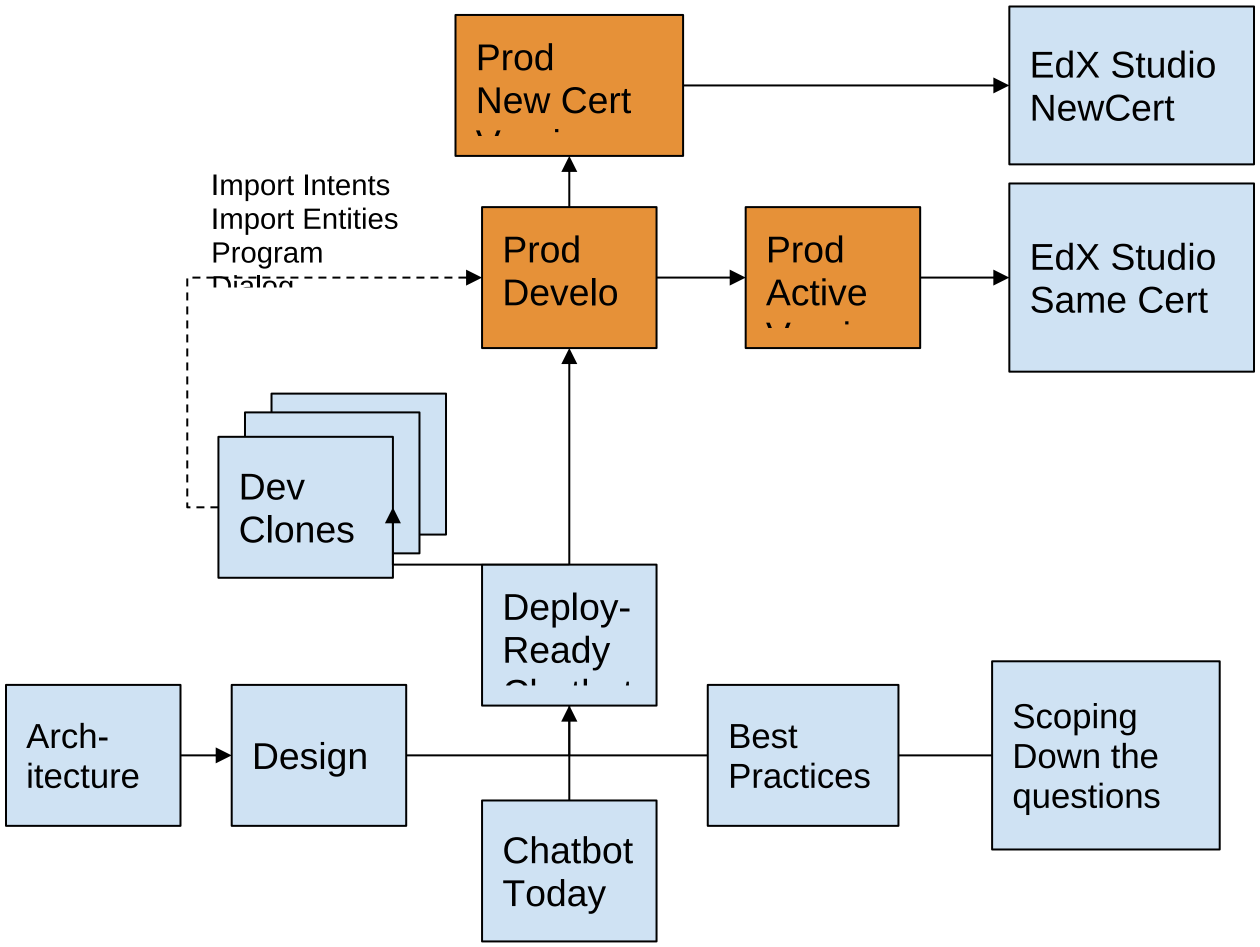
    - Consider what your customers/end users might want to do, and what you want your application to be able to handle on their behalf when you decide on naming intents. For example, you might want your application to help your customers make a purchase. If so, you can add a #buy-something or #Show_Price intent to make a purchase or to get a price of an item.

  - **Entities:** We should correctly identify the data elements that will be addressed in our application and list them all down. For example, if our chatbot is designed to answer questions about a grocery store, then we can list an entity like @grocery_things which can consist of data like milk, bread, eggs, fruits, vegetables etc. Basically stuff that you would expect a customer to ask to a grocery store based chatbot.

- **Grouping:**
  - It is necessary that we know how and which type of intent
- Constraints on development - “think small functional codes”
      - Number of conditions
      - Number of responses
      - Etc.

- Design Chatbot to evolve with course development
  - Program-level information
  - Certificate-level information
  - Course-level information

- Refactor Chatbot to Architecture and Design
  - Should be done by Shishir’s primary backup
  - Tested to completion
- Deploy for testing on Applied Scrum
- Develop extension for new course



- **Version Control:**
  - We will be deploying the chatbot across multiple courses in the UMD MOOC Program.
  - For that, we will maintain a master copy of the chatbot. Then depending on the frequency of reviews of student conversations, which can be daily, weekly or monthly, we will create a clone and integrate all the changes in that clone and will be using that as updated versions of the original one.
  - For creating chatbots for different courses, we will clone the master copy and make specific changes related to course information that is relevant to the course in question and deploy and repeat the same steps above for updating and maintaining different versions for it.