MAi: An Interface for Declarative Specification of Goal-Directed Dialogue Agents

Tathagata Chakraborti · Christian Muise · Shubham Agarwal · Luis A. Lastras · Yasaman Khazaeni
IBM Research AI, USA
{tchakra2, christian.muise, shubham.agarwal}@ibm.com, {lastrasl, yasaman.khazaeni}@us.ibm.com

Abstract

The state of the art of dialogue agents requires a lengthy design process spanning months with experts in the loop who specify complex conversation patterns manually. Our work proposes a paradigm shift in bot design by adopting a declarative approach which composes the full dialog tree automatically. This allows the designer to construct complex dialogue agents from scratch and interact with them in a matter of hours. The demonstration will allow the audience to interact with this new design paradigm and construct their own bots on the spot.

Current enterprise-level goal-directed dialogue agents require significant expertise, time and effort to build (Sreedhar 2018). The process usually requires domain experts to sit down with engineers to construct complex interaction patterns *in the form of explicit dialogue trees*. This process quickly becomes intractable. Existing end-to-end solutions to chat-bots, on the other hand, require little expertise to build (but a lot of data) and offer little to no control over the operational properties of the bot (Metz 2018). As a result, such data driven end-to-end techniques have been predominantly demonstrated in non-goal oriented settings. Especially in the context of design of goal directed dialogue agents for tasks such as customer support, we can conceive of the following desiderata:

- The bot-designer must be in control of the behavior of the bot towards the end-users once it is deployed. This means that the specification of the bot must be interpretable to the designer, and also readily debuggable and editable as desired by the designer.
- The bot designer should be able to easily connect actions available to the bot to actionable system functions and API endpoints in order to get its job done (for example, issue a ticket or place an order).
- Finally, and perhaps most importantly, in order to be able to support complex interaction patterns between the end-user and the bot, the bot designer must be able to completely describe the bot's capabilities without having to manually specify the entire dialog tree.

In the following sections, we describe in detail how our *Model Acquisition Interface* (`MAi`) goes about making these capabilities available to the bot designer by adopting a declarative modeling paradigm. We anticipate that this modeling paradigm will not only reduce the effort in building robust dialogue agents for enterprise use, but also make the design process of such bots more easily accessible to a wider range of users (and thus to a wider range of uses).

Interactions on MAi

Declarative Specification The state of the art for the design of dialogue agents involves manual specification of the entire dialog tree. This process is imperative, in having to enumerate all sequences of possible bot behavior. We instead propose a declarative modeling paradigm built on top of automated planning techniques (Ghallab, Nau, and Traverso 2004) that are uniquely suited declarative modeling – the bot designer starts by putting themselves in the shoes of the agent and conceives of **variables** that the bot needs to keep track of and **actions** that the agent is capable of executing.

Variables These are either `Booleans`, arbitrary `JSON` variables, or variable types defined as *system entities* in Watson Assistant[1] that may be extracted from user utterances. Examples of variables include credit card details (as a `JSON` dictionary), `Boolean` flags determining whether certain pieces of information have been acquired, and so on.

Actions Actions define capabilities of the bot. They operate on the values of the variables defined above towards achievement of its goals. Each action is defined from the perspective of the bot in terms of the information it requires to perform the action and the possible outcomes of performing that action. The structure of an action consists of:

- **Needs** are values – `true` or `false` – or status – `known`, `unknown` or `uncertain` – of variables that need to hold for the bot to be able to perform that action. For example, the credit card details would need to be known before the bot can place an order.
- **Outcomes** define a set of possible events that can occur in response to the bot executing an action – e.g. if the bot calls an API endpoint, it may either receive some data back indicating success or it may receive a failure message. This can be modeled in terms of two different outcomes. When the bot is compiled before deployment, the AI planning techniques in the back-end compose the full dialogue tree while considering all possible outcomes. After deployment, at the time of execution, only one of the outcomes occur when an action is performed – the job of the *execution monitor* is to identify and react to this outcome.
- **Updates** define how the values of different variables change for each outcome. For example, if the user responds with the credit card details, then the values of the variable

[1] `ibm.biz/sys-entities`

corresponding to credit card details are updated, or the status of the order may be updated on after calling an API endpoint as described above. The manner of assignment largely depends on the type of an action, described next.

Action Types `MAi` allows three types of actions that perform different roles during an interaction with the end user. All of them follow the action structure introduced previously, in addition to extra features typical to a particular action type.

- **Dialogue** actions are performed directly with the user via conversation. The designer outlines how the bot expresses itself (such as in asking for credit card details) and possible user utterances in response for each possible outcome.
- **Logic** actions are internal to the bot and are used to maintain state information (for example, in setting credit card details to `known` once the bot determines membership in loyalty program). The designer can specify under what (logical) conditions each outcome occurs and in what order. The ability to design these actions is unique to this design paradigm – internal actions are not accessible in chat logs and are thus inaccessible to end-to-end systems.
- **Cloudfunction** actions allow the bot to respond to actionable items. This is crucial to the development of goal-oriented dialogue agents in domains such as customer support where the agent needs to pull data from a user's account, set information such as usernames, passwords, etc. issue tickets, book orders, and so on. These API endpoints still need to be implemented by a developer – however, `MAi` provides pathways to *simulate* these endpoints in situ so that the designer can chat with the bot immediately.

For dialogue actions, information extracted from the specified user utterances (using classifiers automatically trained using Watson Assistant services) can make updates to variables in each possible outcome. For cloudfunction actions, such updates can come from the response of the API endpoints.

Miscellaneous Features

The Meta-Writer `MAi` incorporates a status bar at the top that provides an (optimistic) estimate of how far along the design process the domain-writer has progressed. This is done with the help of a *meta-writer* that casts the domain writing process itself as a planning problem. This meta-problem models the minimum requirements of a dialogue model given the nature of the domain specification and monitors for fulfillment of those basic requirements as the domain writer is constructing the bot, before the bot can be launched. If there are outstanding items to be modeled, it suggests possible design items to the user who can directly add them into the specification of the bot – for example, if an action needs a particular value of a variable, then there must be some update of some outcome of another action that produces it.

Follow-ups, Confirmations and Slots `MAi` houses special pathways to (1) fill slots (i.e. query for missing information); (2) confirm values of variables when the bot is uncertain; (3) start the conversation with a particular action; (4) end the conversation on a particular outcome; or (5) force the next action as a *follow-up* to a particular outcome. Some of these features hark back to the imperative process of bot design and are left in as syntactic sugar to further streamline the design process as well as make the transition to the declarative modeling paradigm easier.

Deployment

Once the progress bar hits 100%, the designer can be sure that there is *at least one path* towards completion of a dialogue. They can then proceed to build their bot.

- **Generate and Visualize Dialogue Tree** Once a solution is guaranteed, the bot designer can generate the entire dialogue tree and visually inspect it to better understand the dependencies in the domain and identify unmodeled constraints or possible undesired dialogue patterns.
- **Chat with the Bot** The designer can also chat with the bot directly and trace the progress of the dialogue along the visualized tree. This provides a powerful debugging tool towards further refinement of the bot specification.
- **Deploy to Watson Assistant** Finally, the designer can deploy the bot directly to Watson Assistant[2] – this compilation is a good way to appreciate the massive scale-ups achieved in the complexity of the bot with respect to the complexity of the specification.

Supporting AI Technologies

The `MAi` back-end compiles this specification into two forms: (1) an abstraction of the declarative planning process that a blackbox planning can solve, producing the entire dialogue tree; and (2) the configuration that is required for deployment of the dialogue agent (on the executing agent, `Hovor`), including elements such as the example utterances to detect user responses and cloudfunction endpoints. The AI components in `MAi` and `Hovor` are primarily built around technologies from the AI planning community.

`MAi` Back-end

– The planning model of the bot is realized through a *non-deterministic* planner (Muise, McIlraith, and Beck 2012). This allows for modeling of complex dialogue patterns and construction of large dialogue trees in a declarative fashion by being able to describe multiple possible outcomes of an action as described before. Details of this representation are outside the scope of this discussion.

– The meta-writer uses (Ramırez and Geffner 2009) to compile the domain writer's actions on `MAi` into observations that can be compiled into a meta-planning problem, as described before. It uses Fast-Downward (FD) (Helmert 2006) as the underlying planner.

`Hovor` Front-end

As we mentioned before, the non-deterministic planner plans with all possible outcomes of an action while generating the complete dialog tree. However, during execution, only one of these outcomes occur. The realization of an action in implementation is done automatically from the specification in the form of *determiners* which determine which outcome has occurred. For dialogue actions, the determiners rely heavily on natural language processing services from Watson Assistant. `Hovor` orchestrates this entire process and situates the bot in the right location in the dialogue tree. More details of the orchestrator can be found in (Muise et al. 2019).

Attachment *https://ibm.box.com/v/mai-icaps-slides*

[2]https://www.ibm.com/cloud/watson-assistant/

References

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Metz, R. 2018. Microsoft's neo-Nazi sexbot was a great lesson for makers of AI assistants. `https://goo.gl/TF8DQx`. MIT Technology Review.

Muise, C.; Vodolan, M.; Agarwal, S.; Bajgar, O.; and Lastras, L. 2019. Executing Contingent Plans: Challenges in Deploying Artificial Agents. In *AAAI Fall Symposium on Integrating Planning, Diagnosis, and Causal Reasoning*.

Muise, C.; McIlraith, S.; and Beck, C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *ICAPS*.

Ramırez, M., and Geffner, H. 2009. Plan Recognition as Planning. In *IJCAI*.

Sreedhar, K. 2018. What it takes to build enterprise-class chatbots. `https://goo.gl/fRDkDn`. Chatbots.