Emora STDM: A Versatile Framework for Innovative Dialogue System Development

James D. Finch
Department of Computer Science
Emory University
Atlanta, GA, USA
jdfinch@emory.edu

Jinho D. Choi
Department of Computer Science
Emory University
Atlanta, GA, USA
jinho.choi@emory.edu

Abstract

This demo paper presents Emora STDM (State Transition Dialogue Manager), a dialogue system development framework that provides novel workflows for rapid prototyping of chat-based dialogue managers as well as collaborative development of complex interactions. Our framework caters to a wide range of expertise levels by supporting interoperability between two popular approaches, state machine and information state, to dialogue management. Our Natural Language Expression package allows seamless integration of pattern matching, custom NLP modules, and database querying, that makes the workflows much more efficient. As a user study, we adopt this framework to an interdisciplinary undergraduate course where students with both technical and non-technical backgrounds are able to develop creative dialogue managers in a short period of time.

1 Introduction

Constructing a functional end-to-end dialogue system is typically an extensive development process. Depending on the goals, such development often involves defining models for natural language understanding and generation (Section 3), and also creating dialogue management logic to control conversation flow. Training a deep learning-based end-to-end model is a cost-effective way to develop a dialogue agent when the goal is a system conforming to behaviors present in training data; however, substantial development effort must be spent as the developer demands broaden to incorporate features that are not well-represented in available data.

We present Emora STDM (State Transition Dialogue Manager), henceforth E-STDM, a dialogue system development framework that offers a high degree of customizability to experts while preserving a workflow intuitive to non-experts. E-STDM caters to a wide range of technical backgrounds by supporting the interoperability between two popular dialogue management approaches, state machine and information state (Section 4). Our framework makes it easy for not only rapid prototyping of open-domain and task-oriented dialogue systems, but also efficient development of complex dialogue managers that tightly integrate pattern matching, NLP models, and custom logic such as database queries. (Section 5).

2 Related Work

A variety of dialogue development frameworks have emerged to expedite the process of dialogue system creation. These frameworks cater to various use cases and levels of developer expertise. Popular commercial-oriented frameworks are primarily intended for non-experts and have workflows supporting rapid prototyping (Bocklisch et al., 2017). They often allow developers to customize natural language understanding (NLU) modules and perform dialogue management using state machines.

Some frameworks require more expertise, but offer better developer control, by following the information state formulation of dialogue management (Ultes et al., 2017; Jang et al., 2019; Kiefer et al., 2019). According to this formulation, dialogues are driven by iterative application of logical implication rules (Larsson and Traum, 2000). This design provides support for complex interactions, but sacrifices the intuitiveness and development speed of dialogue management based on state machines.

Other frameworks (e.g., ChatScript, botml) rely on custom programming languages to design conversation flow. The custom syntax they specify is based on pattern matching. Although requiring expertise, rapid prototyping in these frameworks is possible with a high degree of developer's control. However, dialogue management focuses primarily on shallow pattern-response pairs, making complex interactions more difficult to model.

ID	Framework	Type	License	SM	IS	PM	IC	EF	ON	ET	CM
1	Emora STDM	Library	Apache 2.0	✓	✓	✓	✓	✓	✓	✓	✓
2	AIML	Language	GNU 3.0			✓					
3	RiveScript	Language	MIT			✓		✓			✓
4	ChatScript	Language	MIT	✓		✓		✓	✓		
5	botml	Language	MIT	✓		✓		✓			
6	OpenDial	Tool	MIT		✓	✓		✓			
7	PyDial	Tool	Apache 2.0		✓			✓	✓		✓
8	VOnDA	Tool	CC BY-NC 4.0		✓	✓		✓	✓		
9	Botpress	Tool	Commercial	✓			✓	✓		✓	
10	RASA	Tool	Commercial	✓			✓	✓		✓	
11	DialogFlow	API	Commercial	✓			✓			✓	

Table 1: Comparison of features supported by various dialogue system development frameworks. SM: state machine, IS: information state, PM: pattern matching for natural language, IC: developer-trained intent classification, EF: external function calls, ON: ontology, ET: error tracking, CM: combine independent dialogue systems.

Table 1 shows a comparison of E-STDM to existing frameworks. E-STDM is most similar to PyOpenDial and botml, which support pattern matching for NLU and tight integration of external function calls. Unlike any existing framework, however, E-STDM explicitly supports both state machine and information state paradigms for dialogue management, and also provides NLU that seamlessly integrates pattern matching and custom modules.[1]

3 NATEX: Natural Language Expression

To address the challenge of understanding user input in natural language, we introduce the NATural language EXpression, NATEX, that defines a comprehensive grammar to match patterns in user input by dynamically compiling to regular expressions. This dynamic compilation enables abstracting away unnecessary verbosity of regular expression syntax, and provides a mechanism to embed function calls to arbitrary Python code. We highlight the following key features of NATEX.

String Matching It offers an elegant syntax for string matching. The following NATEX matches user input such as '*I watched avengers*' or '*I saw Star Wars*' and returns the variable `$MOVIE` with the values '*Avengers*' and '*Star wars*', respectively:

```
[I {watched, saw}
 $MOVIE={Avengers, Star Wars}]
```

The direct translation of this NATEX to a regular expression would be as follows:

```
.*?\bI\b
.*?(?:\b(?:watched)\b|\b(?:saw)\b)
.*?(?P<MOVIE>(?:\b(?:avengers)\b|
                \b(?:star wars)\b)).*?
```

As shown, NATEX is much more succinct and interpretable than its counterpart regular expression.

[1]Emora STDM is available as an open source project at github.com/emora-chat/emora_stdm.

Function Call It supports external function calls. The following NATEX makes a call to the function `#MDB` in Python that returns a set of movie titles:

```
[I {watched, saw} $MOVIE=#MDB()]
```

This function can be implemented in various ways (e.g., database querying, named entity recognition), and its NATEX call matches substrings in the user input to any element of the returned set. Note that not all elements are compiled into the resulting regular expression; only ones that are matched to the user input are compiled so the regular expression can be processed as efficiently as possible.

Ontology It supports ontology editing and querying as the built-in NATEX function called `#ONT`. An ontology can be easily built and loaded in JSON. `#ONT(movie)` in the example below searches for the `movie` node in a customizable ontology represented by a directed acyclic graph and returns a set of movie titles from the subgraph of `movie`:

```
[I {watched, saw} $MOVIE=#ONT(movie)]
```

Response Generation It can be used to generate system responses by randomly selecting one production of each disjunction (surrounded by {}) in a top-down fashion. The following NATEX can generate "*I watched lots of action movies lately*" or "*I watched lots of drama movies recently*", and assign the values of '*action*' and '*drama*' to the variable `$GENRE` respectively:

```
I watched lots of $GENRE={action,
horror, drama} movies {recently, lately}
```

Error Checking Our NATEX compiler uses the Lark parser to automatically detect syntax errors.[2] Additionally, several types of error checking are performed before runtime such as:

[2]https://github.com/lark-parser/lark

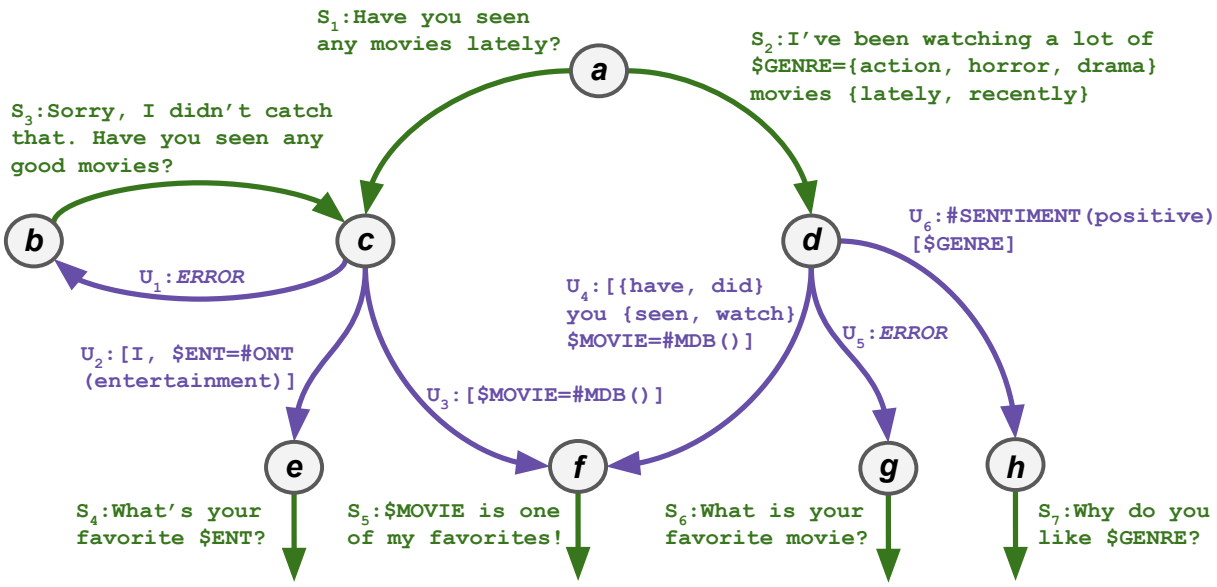

Figure 1: A dialogue graph using a state machine approach with NATEX to dialogue management.

- Call to a non-existing function.
- Exceptions raised by any function.
- Function returning mismatched type.
- Reference to a non-existing variable.

4 Dialogue Management

4.1 Dialogue State Machine

The primary component responsible for dialogue management within E-STDM is a state machine. In our framework, state transitions alternate between the user and the system to track turn taking, and are defined by NATEX (Figure 1). Any transition performed with a variable-capturing NATEX will store a variable-value pair in a dedicated table in memory, which persists globally for future reference.

User turns are modeled by transitions according to which NATEX matches the user input. To resolve cases where multiple NATEX yield matches, transitions can be defined with priority values. Similarly, developers can specify a catch-all "error transition" (ERROR in Figure 1) to handle cases where no transition's NATEX returns a match. The user input resulting in an error transition is automatically logged to improve the ultimate design of the state machine.

System turns are modeled by randomly selecting an outgoing system transition. Random selection promotes uniqueness among dialogue pathways, but can be restricted by specifying explicit priority values. To avoid redundancy when returning to a previously visited state, E-STDM prefers system transitions that have not been taken recently.

The simplicity of this dialogue management formulation allows for rapid development of contextually aware interactions. The following demonstrates the streamlined JSON syntax for specifying transitions S_1, S_3, U_1, U_2, and U_3 in Figure 1.

```
{
  "Have you seen any movies lately?": {
    "state": "c",
    "[I, $ENT=#ONT(entertainment)]": {
      "What's your favorite $ENT?": {..}
    },
    "[$MOVIE=#MDB()]": {
        "$MOVIE is one of my ...": {..}
    }
    "error": {
        "Sorry, I didn't catch ...": "c"
    } } }
```

4.2 Information State Update Rules

Despite its simplicity, state machine-based dialogue management often produces sparsely connected state graphs that are overly rigid for complex interactions (Larsson and Traum, 2000). E-STDM thus allows developers to specify information state update rules to take advantage of the power of information state-based dialogue management.

Information state update rules contain two parts, a precondition and a postcondition. Each user turn before E-STDM's state machine takes a transition, the entire set of update rules is iteratively evaluated with the user input until either a candidate system response is generated or no rule's precondition is satisfied. In the following example, satisfying precondition `[I have $USER_PET=#PET()]` triggers postcondition `#ASSIGN($USER_LIKE=$USER_PET)`

to assign `$USER_PET` to `$USER_LIKE`, allowing rule `#IF(..) I like $USER_LIKE ..` to trigger in turn:

```
{
  "[I have $USER_PET=#PET()]"
  : "#ASSIGN($USER_LIKE=$USER_PET)",
  "[$USER_FAVOR=#PET() is my favorite]"
  : "#ASSIGN($USER_LIKE=$USER_FAVOR)",
  "#IF($USER_LIKE != None)"
  : "I like $USER_LIKE too! (0.5)"
}
```

When a precondition is satisfied, the postcondition is applied through the language generation (Sec. 3). If a real-number priority is provided in parentheses at the end of any NATEX, the generated string becomes a candidate system response. A priority value higher than any outgoing system transition in the dialogue state machine results in the candidate becoming the chosen one; thus, no dialogue state machine transition is taken. Often however, a developer can choose to omit the priority value to use NATEX purely as a state updating mechanism.

This formalism allows flexible interoperability between state machine-based and information state-based dialogue management. Given E-STDM, developers have the latitude to develop a system entirely within one of the two approaches, although we believe a mixed approach lends the best balance of development speed and dialogue sophistication.

4.3 Combining Dialogue Modules

E-STDM has explicit support for a team-oriented workflow, where independent dialogue modules can be easily combined into one composite system. Combining multiple modules requires specification of a unique namespace per module to enforce encapsulation of both errors and identifiers. The following is an example Python script combining dialogue systems `df1` and `df2` under namespaces `DF1` and `DF2`, respectively:

```
df1 = DialogueFlow('start_1')
df1.add_transitions('df1.json')
df2 = DialogueFlow('start_2')
df2.add_transitions('df2.json')

cdf = CompositeDialogueFlow('start')
cdf.add_module(df1, 'DF1')
cdf.add_module(df2, 'DF2')
cdf.add_user_transition(
  'DF1.stateX', 'DF2.stateY',
  "[{film, movie}]")
```

Moreover, inter-component transitions can be made between any two dialogue states to seamlessly combine modules together and allow smooth topic transitions for better user experience.

5 Educational Use of Emora STDM

As an application case study, we present the use of E-STDM in an educational setting. E-STDM is deployed in an interdisciplinary undergraduate course called *Computational Linguistics*,[3] where dialogue system development within E-STDM is a part of the requirements. Students in this course come with varying levels of programming ability; many with little to no imperative programming experience.

Students are tasked with the development of chat-based dialogue systems that can engage a user in 10+ turn conversations. At the time of writing, students have completed two assignments involving dialogue system creation. Students are grouped in teams, with at least one student with prior coding experience per team. Teams are free to select a domain, such as video games, sports, or technology, and are given two weeks for development.

We make the unmodified version of dialogue systems from these students publicly available.[4] The successful use of E-STDM by novice programmers demonstrates the utility of this framework, in terms of its usability and potential as an educational tool.

Acknowledgments

We gratefully acknowledge Sarah E. Finch for her support in developing E-STDM as well as assessing the course assignments (Section 5).

References

T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol. 2017. Rasa: Open source language understanding and dialogue management. *arXiv:1712.05181*.

Y. Jang, J. Lee, J. Park, K. Lee, P. Lison, and K. Kim. 2019. PyOpenDial: A Python-based Domain-Independent Toolkit for Developing Spoken Dialogue Systems with Probabilistic Rules. In *Proceedings of EMNLP System Demonstrations*.

B. Kiefer, A. Welker, and C. Biwer. 2019. Vonda: A framework for ontology-based dialogue management. *arXiv:1910.00340*.

S. Larsson and D. R. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *NLE*, 6(3 & 4):323–340.

S. Ultes, Rojas B., Lina M., P. Su, D. Vandyke, D. Kim, I. Casanueva, P. Budzianowski, N. Mrkšić, T. Wen, M. Gašić, and S. Young. 2017. Pydial: A multi-domain statistical dialogue system toolkit. In *Proceedings of ACL System Demonstrations*.

[3] github.com/emory-courses/cs329

[4] github.com/emora-chat/emora_stdm_zoo