INTERSPEECH 2016: Show & Tell Contribution
September 8–12, 2016, San Francisco, USA

Flexible, Rapid Authoring of Goal-Orientated, Multi-Turn Dialogues Using the Task Completion Platform

Alex Marin, Paul Crook, Omar Zia Khan, Vasiliy Radostev, Khushboo Aggarwal, Ruhi Sarikaya

Microsoft Corp., Redmond, Washington, U.S.A.

alemari,pacrook,omkhan,vasiliyr,khushboa,rusarika@microsoft.com

Abstract

The Task Completion Platform (TCP) is a multi-domain, multi-modal dialogue system that can host and execute large numbers of goal-orientated dialogue tasks. TCP is comprised of a task configuration language, TaskForm, and a task-independent dialogue runtime, allowing task definitions to be decoupled from the global dialogue policy used by the platform to execute the tasks. This separation enables scenario developers to rapidly develop new dialogue systems, by eliminating the need to re-implement the policy from scratch for each new task. In this paper, we introduce support for authoring tasks in a variety of dialogue styles, ranging from entirely flexible to fully system-initiative. This flexibility is enabled by a set of task-level policy override constructs, which augment or constrain the default platform-level policy to achieve the desired system behavior. We demonstrate the use of the TaskForm language to define complex, multi-turn tasks in a variety of domains and add different task-specific policy constructs to demonstrate the flexibility of the task authoring process.

Index Terms: dialog systems, dialog policy

1. Introduction

Dialogue systems are becoming increasingly common in daily use in a variety of settings, such as customer service interactive voice response (IVR) systems, personal digital assistants (PDA) and more recently bots running inside messaging platforms. Such variety of applications necessarily requires a range of authoring styles, from the entirely "on-rails" dialogues used in IVRs, to user-initiative, unconstrained dialogues as is more natural in a PDA or bot setting. At the same time, developing a complete dialogue policy from scratch for each new task or dialogue is resource- and expertise-intensive. The Task Completion Platform (TCP) reduces the effort to author new goal-orientated tasks by providing a task configuration language, *TaskForm*, together with a shared dialogue runtime that applies a system-wide policy to all tasks executed by the system [1]. To enable different authoring styles, we introduce in this paper task-specific policy overrides, allowing task authors to constrain or guide the task execution as necessary while still leveraging the system-wide policy. The remainder of this paper briefly discusses the platform architecture, describes the authoring process, with emphasis on policy overrides, and demonstrates the system capability with a discussion of different tasks.

2. Task Definitions Using System Policy

The TCP runtime has a modular architecture, with the dialogue management process loosely separated into: initial processing of input and natural language understanding (NLU), dialogue state updates, and policy execution. The input to the system may be speech recognition output or typed text. Alternative NLU and dialogue interpretations are preserved in parallel for each step of the processing. Multiple tasks may be executed in parallel, and a final ranking step selects the most likely dialog state and executes the corresponding dialog act. The system is described in more detail in [1].

TCP tasks are defined using the TaskForm language. A TaskForm-specified task that relies entirely on the built-in dialogue policy is encoded as a set of triggers defining under what conditions task execution should begin, a set of parameters defining what information should be collected during the dialogue, and a set of dialog acts defining what is presented to the user. A sample task snippet is shown in figure 1.

A *task trigger* defines the conditions required for the execution of a task to begin. These conditions are represented in terms of NLU results: combinations of domains, intents, and slots (presence or absence), optionally supplemented by must-trigger phrases.

Each *task parameter* defines a container for information required for task execution. The information may be collected directly from the user, inferred during the execution from the values of pre-computed parameters, or mixed. Each parameter definition specifies how the parameter value should be produced and what dialog acts are used to solicit relevant information. The value of a parameter is provided by an associated piece of code, a *resolver*, which is implemented outside the TaskForm.

A *dialog act* captures the information that should be presented to the user when the system selects it. This information includes, at a minimum, a prompt to be read out and a list of strings to be shown on the screen. Allowed parameter dialog acts include: **MissingValue** (ask the user for input required to populate the parameter), **NoResultsFollowUp** (prompt to change information as no results were found), **Disambiguation** (ask the user to select the parameter value from a list), **Confirmation**, and **ConfirmationFailure**.

The default system-level task processing uses the new NLU slots tagged in the user utterance to update all corresponding parameters, according to a topological sort over the inherent parameter dependency graph. Once all the possible updates to parameters have been applied, the system selects a dialog act to present to the user. Preference is given first to task-level dialog acts (such as **Cancellation**, **Confirmation**, or **Completion**), then to dialog acts of parameters for which some user input has already been collected (*e.g.* **Disambiguation** or **NoResultsFollowUp**). If no suitable dialog acts can be produced, the system will select the **MissingValue** dialog act of the next parameter for which no user input has been provided.

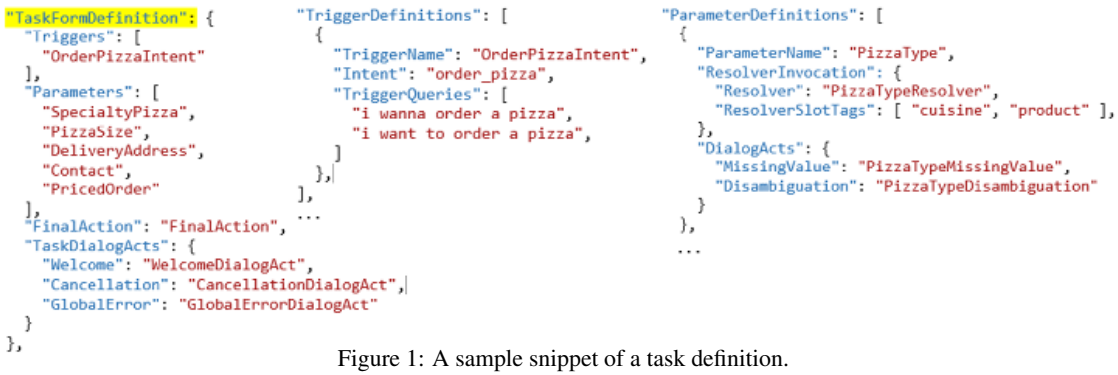

Figure 1: A sample snippet of a task definition.

Figure 2: Example of a validation condition.

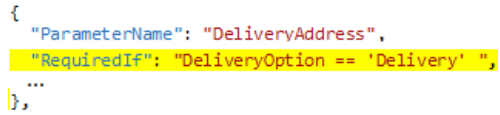

Figure 3: Example of a conditionally-optional parameter.

3. Authoring Policy Overrides

Three TaskForm language constructs are used to define the per-task policy that augment the default system policy.

Validation conditions describe binary (**true**/**false**) conditions which the task must fulfill in order for its execution to be finished. Each validation condition operates over the entire task state and has access to each parameter's value(s) and the state in which the system expects the respective parameter to be (*e.g.* **Filled**, **Empty**, **Resolved**, or **Prompted**). If the validation condition evaluates to **false**, a **ValidationFailure** dialog act is displayed. Validation conditions can be used to implement complex conditions involving multiple parameters (*e.g.* setting up a **MutuallyExclusive** list of parameters), or for implementing more complex conditions that direct task execution given combinations of specific parameter values. Figure 2 shows a validation condition for a timer setting task. If a previous timer is already running, the user is asked whether to reset the existing timer or set a new one.

RequiredIf parameter attributes are constraints indicating that a parameter may be optional, depending on the values of one or more other parameters. In the example of figure 3, the **Location** parameter is required only if the value of the **DeliveryType** parameter is set to **Delivery**. If the **DeliveryType** parameter instead has a value of **CarryOut**, the user's location is not needed and thus the parameter remains optional. The user may still provide a location but will not be prompted for it.

Enhanced dialog acts allow task authors to define fine-grained state update and NLU hints as part of each dialog act. State update hints suggest to the dialog system that the task or its parameters are expected to be in specific states. Task authors can thus override system behavior, *e.g.* to clear out a parameter or force the task to be re-confirmed on a future turn. NLU priming allows various levels of flexibility, from hints to enforcing the tagging of only specific subsets of intents or slots during the next turn. Figure 2 shows an enhanced dialog act. As a result of this dialog act, the task would be marked as **Prompted** for final user confirmation; the expected NLU intents guide NLU tagging for the following turn.

Task-level policy constructs allow for different authoring styles that are similar to those supported by other dialogue management systems. By encoding complex dependency relationships between parameters in RequiredIf blocks, the TaskForms language and TCP allow for an execution model similar to that of RavenClaw [2], in which a hierarchical flow is composed from distinct agents. Validation conditions, NLU constraints, and state update hints can be used to author dialogues akin to those defined in VoiceXML [3], a standard industry tool used to build system-initiative IVRs, in which the understanding of user input is constrained at each turn, with no opportunity for user initiative.

4. Demo Outline

As a starting point we will use three tasks of broad applicability in the personal assistant space: *food ordering* (scoped to ordering pizza for simplicity), *restaurant reservation*, and *timer setting*. A section of the food ordering TaskForm is shown in figure 1. We will demonstrate the execution of the tasks with and without task-specific policy definitions, highlighting how task authors can fine-tune user interactions in specific cases while relying on the platform policy for most of the task execution. In particular, we will show the pizza ordering task with and without conditioning the **Location** parameter on the value of the **DeliveryType** parameter, and the timer task with and without the added validation condition or enhanced dialog acts. Some pre-recorded demo videos can be found at http://research.microsoft.com/en-us/people/pacrook .

5. References

[1] P. A. Crook et al., "Task Completion Platform: A self-serve multi-domain goal-oriented dialogue platform," in *Proc. NAACL*, 2016.

[2] D. Bohus and A. Rudnicky, "The RavenClaw dialog management framework: Architecture and systems," *Computer Speech and Language*, 2009.

[3] VoiceXML. (2000) VoiceXML version 1.0. https://www.w3.org/TR/voicexml/.