I Can Parse You: Grammars for Dialogs

Martin Hirzel[1], Louis Mandel[2], Avraham Shinnar[3], Jérôme Siméon[4], and Mandana Vaziri[5]

1 IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
hirzel@us.ibm.com
2 IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
lmandel@us.ibm.com
3 IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
shinnar@us.ibm.com
4 IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
simeon@us.ibm.com
5 IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
mvaziri@us.ibm.com

Abstract

Humans and computers increasingly converse via natural language. Those conversations are moving from today's simple question answering and command-and-control to more complex dialogs. Developers must specify those dialogs. This paper explores how to assist developers in this specification. We map out the staggering variety of applications for human-computer dialogs and distill it into a catalog of flow patterns. Based on that, we articulate the requirements for dialog programming models and offer our vision for satisfying these requirements using grammars. If our approach catches on, computers will soon parse you to better assist you in your daily life.

1998 ACM Subject Classification I.2.7 [Natural Language Processing] Discourse

Keywords and phrases Bots, virtual agents, dialog managers, domain-specific languages

Digital Object Identifier 10.4230/LIPIcs.SNAPL.2017.6

1 Introduction

This paper is about authoring systems that allow dialogs between humans and computers. When humans can converse with computers using natural language, computers can assist them in real-life situations where traditional human-computer interfaces are cumbersome. Recent advances in natural language processing paved the way to bring conversational human-computer interfaces to the mainstream. However, facilities for authoring such interfaces are lagging behind. It is not easy to program a robust yet powerful human-computer dialog. On top of that, there is a confusing variety of use-cases for conversational interfaces. This paper takes a stab at organizing these use cases by cataloging recurring conversational flow patterns. Based on that, this paper outlines a vision for a new programming model in which developers specify human-computer dialogs via grammars.

Commercially successful prior approaches for specifying human-computer dialogs are the finite-state approach and the frame-based approach [13, 19]. In the finite-state approach, dialog control is determined by an explicitly-specified directed graph [12, 18], whereas in the frame-based approach, the dialog is driven by filling slots in a form [3, 18]. Unfortunately, both approaches suffer from what we term the *coherence-flexibility dilemma*: a dialog should be both coherent (yield the right outcome) and flexible (adapt to the human), but these goals conflict. For instance, coherence can be maximized via confining prompts and explicit confirmations, but those can annoy humans or even keep them from reaching their goals.

Besides elaborating on the above problem statement, this paper also offers a vision towards a new programming model that solves it. First, we observe that the human utterances in a dialog constitute a linear sequences of inputs. Authoring a dialog means imposing structure over that sequence. A well-known and successful formalism that does exactly that is grammars. The idea is that the conversational agent is like a parser, and the natural-language understanding is like a lexer in a compiler. Continuing with the analogy, the outcome of the conversation is like an abstract syntax tree. The programming language community has plenty of expertise in using grammars to specify parsers along with their outcomes. Here, we put a twist to that by turning entire utterances by the human in a dialog into individual tokens in a grammar.

Overall, human-computer dialogs are a hot trend in computing. There is a lack of programming models for authoring them. This paper gives an overview of this trend and outlines a vision for a new programming model for it.

2 Why Human-Computer Dialog

Natural-language human-computer dialogs have been a mainstay of science fiction for a long time. In fiction, humans and computers or robots can carry wide-ranging conversations and interact almost as equals. Given this appeal, it is not surprising that fact follows fiction, although we must obviously keep reasonable expectations for how natural this interaction is [19]. Many early dialog systems focused on travel, where a human who is away from home can use a good old voice phone to book, for instance, flights [13].

Recently, human-computer dialog systems, also known as chat bots or virtual agents, received much renewed attention [14]. Cutting through the hype, there are several good reasons for that. One is that common-place devices such as cars, thermostats, or watches are becoming "smart", but we typically do not want large displays on them to interact via graphical user interfaces. Another reason is that even on laptops or smart-phones with adequate displays, we do not always want graphical interfaces with many screens and clicks. For instance, when using a messaging platform such as Slack, it can be preferable to interact right there with bots, rather than context-switching to a different application. Such integrated interaction also benefits from other messaging features such as history. Another reason is that when computers understand the way humans speak, humans need not adopt a form of "machine-speak".

Given this trend, several companies are starting to offer services that let customers author and run their own chat bots. For instance, the authors of this paper became involved in this trend because their employer started offering the Watson Conversation Service (WCS) [12]. WCS and its competitors can be put to immediate use to augment existing websites, messaging platforms, or mobile apps with a conversational interface. Furthermore, they are already being used for dialogs with robots. In the not-so-distant future, computers will enhance human cognition and ultimately work side-by-side with humans. When automated cognitive assistants become rich in features and reason in human concepts, they will eventually reach an inflection point where non-conversational interfaces no longer suffice to interact with them.

Ultimately, natural-language interfaces take advantage of our biology. Large areas of the human brain are devoted to natural-language communication [11]. It is easy for humans to use speech in circumstances that prevent using graphical user interfaces, such as in the dark, or hands-free (e.g. while cooking), or eyes-free (e.g. while driving). In fact, natural-language interfaces make computing technologies accessible to parts of the population that were previously excluded, such as small children, elderly, or visually impaired people.

Now that we have motivated why humans should converse with computers using natural language, let us explore what kinds of conversations they may have.

3 Use Cases and Flow Patterns

Use cases for human-computer dialogs abound in a staggering variety of domains, from travel to retail, entertainment to medical to automotive to technology trouble-shooting, and beyond. At first glance, it would seem like dialogs in each of these domains look very different. Yet as someone setting out to provide dialog authoring environments, we wanted to ensure we understand and capture the common patterns. To this end, this section contributes a catalog of flow patterns for bots. The catalog maps out the terrain, gives structure, and establishes terminology. Such an overview is a prerequisite to prioritization so we can arrive at the right scope of what to focus on and what to leave out.

A *flow pattern* is an interaction of a few back-and-forth turns in a dialog that either has a single well-defined outcome or handles a single deviation from the primary purpose of the bot. People with a programming-languages background can think of the *outcome* of a conversation as a data structure that can serve as a parameter to an external service call or simply as a record of what happened in the conversation. The outcome data structure fits the type for all conversations by that particular (part of a) bot. A flow pattern is finer-grained and lower-level than a *use case*, which captures an entire conversational agent for a particular purpose that may involve several flow patterns. In other words, flow patterns are intended to be domain-independent and indeed occur across many use cases in many domains. We distinguish two kinds of flow patterns: *outcome-driven patterns*, where the back-and-forth of conversation is directed at producing an agreed-upon outcome, and *add-on patterns*, which can occur during outcome-driven patterns but delay or possibly even derail the outcome. Another way to think of it is that outcome-driven patterns are about coherence whereas add-on patterns are about flexibility.

We conducted an informal survey and interviews with assorted product and client teams at IBM, and distilled our learning into the following set of flow patterns:

Outcome-driven flow patterns: Question answering, command and control, form filling, diagnosis, problem resolution, query formulation.

Add-on flow patterns: Anaphora and ellipsis, repair, disambiguation, elaboration, stopping a discussion, digression, interleaved dialogs.

Below, we elaborate on each of these patterns with brief definitions and discussions as well as example dialogs between a human (H) and a computer (C). Beyond this basic list, one can also consider bot features such as chit-chat, augmenting the bot with multi-modal input, vision, or touch, or making it empathetic by detecting and appropriately responding to the emotions of the human. To keep this paper focused, we elected not to elaborate on those other features here.

3.1 Outcome-Driven Flow Patterns

Question-answering flow pattern

Automatically answer questions posed by human in natural language based on a corpus of documents. From the perspective of the dialog flow, this is one of the simplest patterns.

One challenge is that questions often depend on context, such as the location. But the main challenge comes from dealing effectively with large numbers of questions. One issue is keeping the dialog in sync with the corpus. Another issue is that distinguishing between similar questions challenges current natural-language understanding technologies.

H: Where is the gift shop?
C: In the front lobby near the elevator.

Figure 1 Example for question-answering flow pattern in hotel concierge use-case.

Command-and-control flow pattern

Control a device through voice or natural language (e.g., radio, phone, thermostat, coffee machine, wheelchair). From the perspective of the dialog flow, this pattern is also simple. One challenge is detecting the conversation start, typically handled by anthropomorphizing the bot with a name such as "Olli" in the example. Another challenge is the integration with the device, which necessitates calling out from the dialog to external service code. And a further-reaching challenge is safety: if the device being controlled performs more serious actions than playing music, it is more important to ensure the human was understood correctly.

H: Olli, play some jazz music.
C: Okay, playing WBGO Jazz 88.3 FM.

Figure 2 Example for command-and-control flow pattern in car dashboard use-case.

Form-filling flow pattern

For each slot in a form, ask the human for the information to fill the slot. The dialog should catch mistakes without being too pedantic. One challenge is to allow the human to control the order in which slots get filled. Another challenge is that a single human utterance may contain multiple pieces of information. Doing this naturally in a finite-state approach is cumbersome. On the other hand, frame-based dialog interpreters are custom-tailored to this pattern [3, 18].

C: Welcome to the pizza dialog! What would you like?
H: A salami and mushrooms pizza.
C: Okay, salami and mushrooms. Any other toppings?
H: No thanks.
C: What size?
H: Medium, please.

Figure 3 Example for form-filling flow pattern in pizza-ordering use case.

Diagnosis flow pattern

Ask the human diagnostic questions following a decision tree to reach a leaf with a classification. The computer may be able to omit some questions if it can obtain the answers from context. One challenge is that the human may retract an earlier answer, either because they were uncertain or because they want to explore alternatives, thus requiring the bot to back-track. Another challenge is that questions posed by the computer are typically hand-crafted and cannot simply be derived from the name of a field in the outcome.

C: Welcome to the pizza dialog! What would you like?
H: A salami and mushrooms pizza.
C: Okay, salami and mushrooms. Any other toppings?
H: No thanks.
C: What size?
H: Medium, please.

Figure 4 Example for form-filling flow pattern in pizza-ordering use case.

Problem-resolution flow pattern

After diagnosis, walk human through how to fix their issue. The main challenge here is to actually come up with good instructions for fixing common issues, and refine the conversational agent so humans have success with it. Also, as in diagnosis, this may involve context and back-tracking.

C: Turn on the projector and connect your video cable.
H: Okay, it is on, but not showing my display!
C: Hit Fn-F7 on your laptop.
H: Good, now I can see my display. Thanks!

Figure 5 Example for problem-resolution flow pattern in technology trouble-shooting use case.

Query-formulation flow pattern

Query information in a database or knowledge base. One of the challenges with this pattern is that it is meta-data specific: human utterances mention pieces of the database schema. A related challenge is to enforce valid queries. Finally, if the human refines the query over the course of the dialog, they tend to use anaphora and ellipsis.

H: What was the median salary for working women in the US in 2009?
C: The median weekly earnings for women in 2009 was 657 US$.
H: Give me that number per race and ethnicity.
C: Here are the corresponding numbers: [...]

Figure 6 Example for query-formulation flow pattern in census use case.

3.2 Add-On Flow Patterns

Anaphora and ellipsis flow pattern

In linguistics, an anaphora is a reference, for instance, "it" in Figure 7 Line 3. An ellipsis is an omitted clause, for instance, the implied "from Croton-Harmon to Grand Central" in the last human (H) utterance in Figure 7 Line 5. Both anaphora and ellipsis must be understood from context. Furthermore, they involve linguistic subtleties that make natural-language understanding more challenging.

H: When is the next train from Croton-Harmon to Grand Central?
C: At 9:03am.
H: Is it peak or off-peak?
C: That's a peak-hour train.
H: When is the next off-peak train?
C: The next off-peak train from Croton-Harmon to Grand Central is at 9:28am.

Figure 7 Example for anaphora and ellipsis flow pattern in travel use case.

Repair flow pattern

Repair describes how parties in a conversation deal with problems in speaking, hearing, or understanding. These problems can occur both when uttering something and when receiving it, for both human and computer. Challenges include detecting the need for repair and wrapping up the repair in a small number of conversation turns, yet avoiding a repeat of the misunderstanding.

H: How much is full total coverage?
C: What kind of coverage? The choices are comprehensive or collision.
H: comprehensive
C: For your car, comprehensive is $983.70.
H: I meant collision.
C: For your car, collision is $685.40.

Figure 8 Example for repair flow pattern in insurance use case.

Disambiguation flow pattern

Agree upon one among alternative meanings of a word or phrase. As for repair, one challenge is detecting the need for disambiguation in the first place. Another challenge is to predict the likelihood that the default choice is good enough, to make a judgment call whether disambiguation is worth the effort.

H: Schedule a meeting with Alice at 10am.
C: Alice is in Texas. Do you mean 10am Central Time?
H: No, I meant 10am Eastern.

Figure 9 Example for disambiguation flow pattern in calendaring use case.

Elaboration flow pattern

Let the human obtain more information on the same topic. The challenge is to anticipate what the human might want to elaborate on, and to keep the system tractable despite the possible explosion of topics.

H: What's the weather tomorrow?
C: 30 degrees with snow.
H: How much snow?
C: 18 to 24 inches. Better stay at home!

Figure 10 Example for elaboration flow pattern in weather use case.

Stopping a discussion flow pattern

Let the human interrupt a conversation or a sub-dialog. One challenge is to decide on the right depth of interruption, i.e., how far to unwind nested levels of subdialogs. Another challenge is that the user might later change their mind about stopping the discussion, and want to resume it again.

C: Do you take anything in your coffee?
H: Skim milk.
C: I'm sorry, we're out of skim milk.
H: Then I don't want coffee.
C: Is there anything else I can get for you instead?

Figure 11 Example for stopping a discussion flow pattern in barista use case.

Digression flow pattern

Allow the human to take the conversation to a different, unrelated topic, and resume the prior topic afterwards. This requires the computer to keep the context of the main thread of conversation, and to re-establish the context for the human when switching back. A variant of digression is temporarily switching to a different part of the same dialog.

C: What's the algorithmic complexity of inserting an item in a B-tree?
H: What are B-trees used for?
C: B-trees are used when not all of the data fits in memory, for instance, in databases. What's the algorithmic complexity?
H: Hm, I guess logarithmic?

Figure 12 Example for digression flow pattern in tutoring use case.

Interleaved dialogs flow pattern

Conduct two or more dialogs concurrently. As in the example, this pattern comes up in use-cases where the conversation is about a situation that unfolds slowly in the real world. Beyond the challenges of digression, this requires keeping multiple contexts active. This makes it more challenge for both participants of the conversation to follow which dialog an utterance belongs to.

H: Correct course to the South.
C: How many degrees?
H: Where is the other vessel headed?
C: 20 degrees Northwest.
H: Okay, correct course 15 degrees South.
C: Initiating course correction.
C: The other vessel is no longer on collision course.

Figure 13 Example for interleaved dialogs flow pattern in naval use case.

The following section describes the most important add-on flow patterns from a different perspective that hints at a solution for them.

C: What size?
H: Mega-sized!
C: Okay, medium. What crust?
H: Uhm, I didn't say medium.
C: Okay, correcting the size. The choices are small, medium, or large. What size?
H: Large.
C: You picked large size. Is that correct?
H: Yes.
C: What crust?

Figure 14 Example dialog with correction.

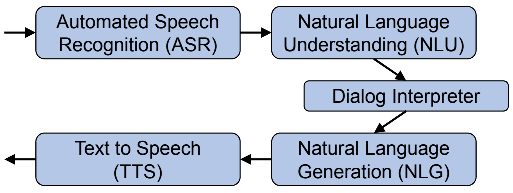
Automated Speech Recognition (ASR)
Natural Language Understanding (NLU)
Dialog Interpreter
Natural Language Generation (NLG)
Text to Speech (TTS)

Figure 15 Basic chat bot architecture.

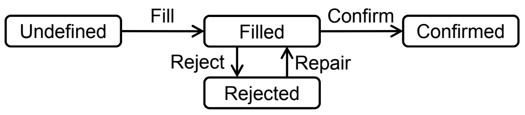

Figure 16 Outcome coherence state machine.

4 Coherence-Flexibility Dilemma

A dialog has *coherence* if its participants have common ground, i.e., they reach mutual belief that contributions have been understood sufficiently well for current purposes [6]. Coherence is essential to reaching the desired outcome of the dialog. Unfortunately, today's natural-language understanding (NLU) technologies have limitations that can put coherence at risk. When that happens, the add-on flow patterns repair and disambiguation from Section 3.2 are crucial.

A dialog has *flexibility* if its participants have both perceived and real control over it. If the dialog has too little flexibility, humans find it difficult to use, since they essentially have to learn its "machine-speak". Ultimately, humans will get frustrated and seek alternative communication channels. Flexibility relates to the add-on flow patterns elaboration, stopping a discussion, and digression from Section 3.2.

The *coherence-flexibility dilemma* is that these two goals are diametrically opposed. It is easy to accomplish one while ignoring the other. But techniques that improve coherence reduce flexibility and vice versa. Some of the early IVR (interactive voice response) systems were so inflexible that people referred to them as phone jail, trying to escape by immediately demanding a human operator.

The good news is that this dilemma is not unique to human-computer dialog. Coherence and flexibility are essential to human-human dialog as well, and human-human dialog has natural and effective ways to balance them. For instance, Clark and Schaefer argue that each utterance has two purposes: first, a backward-looking confirmation of the previous utterance, and second, a forward-looking question or statement advancing the conversation [7]. There is flexibility in how implicit or explicit the confirmation is, and humans adjust their style when they detect misunderstandings.

Consider for example the dialog in Figure 14. When the computer says "Okay, medium. What crust?", it attempts an implicit confirmation of what it understood (backward-looking) and asks the next question (forward-looking). The human corrects the computer. Next, the computer rephrases the question for the size by explicitly listing the choices, thus sacrificing some flexibility to improve coherence. After the human picks an option, the computer conducts a more explicit confirmation before going back to business as usual.

The good thing about this natural confirmation mechanism from human-human dialog is that it reduces the burden on the NLU technology. Instead of waiting for computers to get better at understanding natural language, we can work with the limitations of current technology if we are prepared to handle the occasional misunderstanding. And even if computers reach human parity in conversational NLU, that still does not imply zero errors, so repair capabilities remain necessary.

Figure 15 shows a simplified architecture for conversational agents, based on literature surveys [13, 19]. Human speech first gets converted to text, and then NLU extracts relevant inputs for the dialog interpreter. Symmetrically, the outputs from the dialog interpreter first get converted to text, and then synthesized back to speech. If the human interacts at the textual level, the speech components can be omitted from the architecture. While there is work on sophisticated NLU that understands parts-of-speech etc., this is brittle when human utterances defy grammar and is harder to port between different natural languages. Therefore, recent systems adopt a simpler and more robust approach to NLU, which merely extracts *intents* and *entities* from the human utterance [12, 23]. An intent is something like “turn on radio”, and an entity is something like “jazz music”. Intents can be detected via machine-learning classifiers, and entities via pattern-matching.

For coherence, we propose that each piece of the outcome data structure be subjected to the state machine in Figure 16. When NLU extracts an intent or entity, that can be used to fill a slot. But being merely filled is not enough. The computer gives the human an opportunity to confirm or reject a slot before it considers it part of the common ground. For flexibility, we propose enabling humans to take the *initiative* when they want to. The most operable definition of initiative we found comes from Derek Bridge, who simply says it belongs to whoever contributes the first part of a conversational adjacency pair [4].

Now that we have established the kinds of dialogs we want to enable, we will get back to the question of how to author them.

5 Grammars to the Rescue

Let us briefly summarize the requirements for a programming model for conversational agents. The agent needs to conduct a linear sequence of interactions with a human over time, consisting of utterances in a conversation. From this sequence of interactions, the agent needs to construct an outcome that adheres to a known type. However, it must detect and fix misunderstandings and allow the human to go off-script by grabbing the initiative where appropriate. Finally, the programming model should be easy to learn, ideally reusing widely-known, familiar, and well-understood programming-language concepts.

We hypothesize that grammars address these requirements well. Grammars specify parsers that process a linear sequence of tokens, produce an outcome, and can be made robust to certain kinds of errors. Most computer science students learn about grammars early in their education, so the concepts should be familiar to them. As an added benefit, in many cases, the outcome of a conversation gets transformed into a command or query for another system, which is itself also best characterized by a grammar. Therefore, our vision is to use grammars as a programming model for conversational agents. Finally, grammars are naturally compositional, thus facilitating modularity and reuse in dialog specification.

To be clear, in our vision, the grammar operates over tokens at the granularity of entire utterances by a human. In other words, we are not concerned with using grammars to parse an individual natural-language sentence, unlike some prior work [5, 20]. In terms of the architecture in Figure 15, we are proposing grammars to specify the dialog interpreter, not other components such as the NLU. Circling back to the paper title (“I can parse you”), we are envisioning a chat-bot as a parser for its human interlocutor, and we are thinking of the NLU component as merely the lexer that extracts tokens in the form of intents and entities from human utterances.

```
query         : select from where?;

select        : selectExpr+;
selectExpr    : selectColumn | selectAll | selectAs;
selectColumn  : columnName;
selectAll     : "all" / "*" / "all␣columns" / "star";
selectAs      : expression columnName;

from          : tableName+;
where         : condition;

condition     : andCondition | orCondition | eqCondition;
andCondition  : condition and condition;
and           : "and" / "conjunction";
orCondition   : condition or condition;
or            : "or" / "disjunction";
eqCondition   : expression eq expression;
eq            : "equals" / "=" / "is" / "equality" / "the␣same␣as";

expression    : STRING | INT | columnExpr;
columnExpr    : columnName;

tableName     : TABLE;
columnName    : COLUMN;
```

Figure 17 Grammar that guides database query dialog.

$$\begin{tikzcd}
\text{Grammar} \arrow[r, "\text{translates to}"] \arrow[d, "\text{guides}"'] & \text{Type} \arrow[d, "\text{validates}"] \\
\text{Dialog flow} \arrow[r, "\text{produces}"'] & \text{Outcome}
\end{tikzcd}$$

Figure 18 Commuting diagram. For examples see Figures 17 (grammar), 19 (dialog), 21 (outcome), and 20 (type).

We are still in the early stages of developing our programming model, but to make things more concrete, Figure 17 offers an example. The grammar guides a dialog that allows the human to issue a database query. It is based on a SQL subset and would be easy to extend for additional clauses (e.g., order-by) and expressions (e.g., arithmetic). As is customary in databases, the grammar distinguishes conditions from expressions, thus enforcing syntactically where one or the other can occur. By design, most of the grammar will look familiar to readers, with non-terminals (e.g., *query*), literal terminals (e.g., `"all"`), multi-valued terminals (e.g., *TABLE*), options (?), repetition (+ or `*`), choice (|), and products (e.g., *select from where*?). One new feature is synonyms (/), for instance, `"and"/"conjunction"`. Synonyms are viewed as samples for NLU tokens, and it is up to the implementation how literally to take them. In some cases, they just serve as training inputs for a machine-learning classifier. As one would expect from non-trivial grammars, this grammar includes nesting (*expression* within *selectAs* within *query*) and even recursion (an *andCondition* can be nested as a *condition* within another *andCondition*).

A single grammar such as the one in Figure 17 both guides the flow of the conversation and specifies the type of its outcome. As illustrated by the commuting diagram in Figure 18,

1 C: Which table do you want to query?
2 H: The Customers table please.
3 C: Okay, Customers. Any other tables?
4 H: Nope.
5 C: Is there a condition?
6 H: Yes: the country is "US".
7 C: Okay, the country equals "US". Any other conditions?
8 H: Yes: the zipcode is 55555.
9 C: Okay, the zipcode equals 55555. Any other conditions?
10 H: Nope.
11 C: What should the query return?
12 H: What are the available columns?
13 C: The available columns are: name, age, address, and zipcode.
14 H: The name
15 C: Ok, the name. Anything else being returned?
16 H: The address.
17 C: Ok, the address. Anything else being returned?
18 H: No, that's all.
19 C: Here is the query:

```
SELECT name, address
FROM Customers
WHERE country = "US" and zipcode = 55555;
```

Would you like to issue it?
20 H: Yes please.

Figure 19 Dialog for database query based on grammar in Figure 17.

the grammar guides the dialog flow. The grammar does not and should not explicitly specify every last detail of the dialog flow. Keeping the flow somewhat under-specified gives the dialog interpreter room to adapt, for instance, by making confirmations more or less explicit depending on whether there are many or few repairs. When the dialog interpreter uses the grammar to implement the dialog flow, it produces an outcome. The outcome could be represented as a JavaScript Object Notation (JSON) document. We implemented a translator from dialog grammars to TypeScript types that validate the final outcome. A coherent dialog yields a valid outcome, but a flexible dialog populates it in the order and style preferred by the human user.

Figure 19 shows a mock-up example dialog driven by the grammar in Figure 17. It occupies the lower left corner of Figure 18. On Line 1, the computer prompts for a table (to fill the *from* factor of the *query* product). On Line 2, the human gives a response in colloquial natural language, from which the NLU extracts a *TABLE* token with the value *Customers*. On Line 3, the computer first echoes back the table name (to establish common ground) and then asks whether there are other tables (implementing the repetition *tableName+* in the grammar). On Line 4, the human says "Nope", from which the NLU extracts an intent of "no". On Line 5, the computer asks whether there is a condition (implementing the option *where?* in the grammar). On Line 6, the NLU can extract multiple tokens from a single human utterance: an intent of "yes", a *COLUMN* token "country", a synonym "is" for "equals", and a *STRING* token "US". The conversation continues to flow as specified in the grammar. Note that while the grammar specifies the select clause first, the order can be re-arranged to get a more natural dialog from a user standpoint, in this example by asking first about the SQL from clause rather than the select clause.

We have not yet implemented our ideas to support the example in Figure 19. Besides illustrating the aspiration of grammar-driven dialog flow, the example also illustrates additional desirable features for which we have yet to design integration points. For instance, on Line 12, the human goes off-script by requesting help. The computer replies with context-sensitive help driven by the underlying database schema. Later, on Line 19, the computer prints the outcome of the conversation using SQL syntax. Assuming that the raw outcome is a JSON document, this would require a simple pretty-printer. Both the schema-awareness and the pretty-printing are technically feasible but not specified by the grammar.

Figure 20 shows the TypeScript type for outcomes of our running example, and Figure 21 shows the concrete JSON outcome. The type and outcome occupy the top and bottom right corners of the commuting diagram in Figure 18. The type figure corresponds line-for-line to the grammar. For instance, Line 9 of the grammar, *from* : *tableName*+;, maps directly to Line 9 of the type, `type FROM = { tableNames: TABLENAME[ ]; }`. The JSON outcome in Figure 21 is essentially an abstract syntax tree for the SQL query in Figure 19 Line 19.

As we are embarking on the project to make this vision reality, it is healthy to also specify some success criteria. These can be found in the next section.

6 How Will we Know it Works?

When building a programming model for conversational agents, we aim at productivity for the dialog authors and quality for the actual human-computer dialogs. In a business context, the former decreases cost and the latter increases revenue. While monetary cost and revenue are concrete numbers, they will only become apparent when the system gets adopted. In the meantime, we need shorter-term metrics to guide our research.

For developer productivity, we are getting guidance from two sources. One is that, being in an industrial research lab, we have access to product and solution teams. We frequently solicit their opinion on the programming model we are proposing as our design is under way. Another source of guidance is to drive the development with several example programs of our own. We turned each flow pattern from Section 3.1 into a test case for our new programming model. In addition, we are extracting the essence from several customer use cases into test cases as well. The authors have had their share of experiences with programming language designs, some of which shipped in IBM products (e.g. [21]). Finally, some readers with a programming language background may appreciate the crazy idea of making our programming model meta-circular, by specifying a dialog whose outcome is the specification for another dialog.

For dialog quality, one might be tempted to adopt metrics from non-dialog tasks such as machine translation. Unfortunately, recent work demonstrates that these correlate very weakly with human judgment [17]. Such metrics are more appropriate to other components of the architecture in Figure 15 such as NLU, or to the simplest flow patterns such as question answering or command-and-control. Instead, a seminal paper on evaluating dialogs proposes ParaDiSE (Paradigm for Dialog System Evaluation) [22]. ParaDiSE postulates that the goal of a dialog is to maximize task success and minimize cost. In our terms, task success consists of producing the outcome the human wanted, and metrics for cost include the number of utterances, repair ratio, etc.

```
type TOP = { query: QUERY; };
type QUERY = { select: SELECT; from: FROM; where?: WHERE; };

type SELECT = { selectExprs: SELECTEXPR[ ]; };
type SELECTEXPR = SELECTCOLUMN | SELECTALL | SELECTAS;
type SELECTCOLUMN = { columnName: COLUMNNAME; };
type SELECTALL = "*";
type SELECTAS = { expression: EXPRESSION; columnName: COLUMNNAME; };

type FROM = { tableNames: TABLENAME[ ]; };
type WHERE = { condition: CONDITION; };

type CONDITION = ANDCONDITION | ORCONDITION | EQCONDITION;
type ANDCONDITION = { condition: CONDITION; and: AND; condition1: CONDITION; };
type AND = "and";
type ORCONDITION = { condition: CONDITION; or: OR; condition1: CONDITION; };
type OR = "or";
type EQCONDITION = { expression: EXPRESSION; eq: EQ; expression1: EXPRESSION; };
type EQ = "=";

type EXPRESSION = string | number | COLUMNEXPR;
type COLUMNEXPR = { columnName: COLUMNNAME; };

type TABLENAME = string;
type COLUMNNAME = string;
```

Figure 20 Type for outcomes of database query dialogs based on grammar in Figure 17.

```
{ query:
  { select : {
      selectExprs: [{ columnName: "name" }, { columnName: "address" }]
    },
    from: { tableNames: ["Customers"] },
    where: {
      condition: {
        condition: {
          expression: { columnName: "country" },
          eq: "=",
          expression1 : "US"
        },
        and: "and",
        condition1 : {
          expression: { columnName: "zipcode" },
          eq: "=",
          expression1 : 55555
        }
      }
    }
  }
}
```

Figure 21 Outcome of database query dialog in Figure 19.

7 Related Work

We found little prior work on using grammars or types to specify dialogs. Bridge beautifully codifies the essence of polite conversation into a grammar [4], but in addition, requires a separate task model to specify a complete chat-bot, whereas we use a grammar to specify the chat-bot itself. GF is a framework for describing grammars of natural languages [20], but focuses on the NLU and NLG components of the architecture in Figure 15, whereas we focus on the dialog interpreter. Bringert uses GF for dialogs [5], but requires the developer to specify multiple grammars and use dependent types, whereas our programming model uses a single grammar and is simpler. Finally, Denecke and Weibel propose a type system for dialogs that constrains individual slots in a frame [8], whereas our types specify the entire dialog flow and validate its outcome.

McTear wrote a survey about spoken dialog technology [19] and the text book by Jurafsky and Martin contains a chapter about dialog and conversational agents [13]. While neither of them focuses on programming models for dialogs, they describe different approaches, two of which imply simple programming models. First, the finite-state approach scripts the dialog at a low level, making it powerful but cumbersome [12, 23]. Second, the frame-based approach focuses on the form-filling pattern [3, 18]. One way to view our vision for a grammar-based dialog programming model is as a generalization of frames.

Other work on programming models for natural language includes natural-language interfaces to databases [1]; CNL (controlled natural language) [2, 9, 16]; and synthesis from natural language [10, 15]. Dialog can be viewed as an alternative that is less controlled than CNL but has a broader scope than synthesis.

8 Conclusion

This paper took a programming-language audience on a tour of the exciting trend of natural-language dialogs between humans and computers. We collected a catalog of relevant flow patterns and articulated the coherence-flexibility dilemma, which we believe to be the key to building good conversational agents. This paper also briefly described our vision of using grammars as a programming model for dialogs.

References

1 Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

2 Matthew Arnold, David Grove, Benjamin Herta, Michael Hind, Martin Hirzel, Arun Iyengar, Louis Mandel, V.A. Saraswat, Avraham Shinnar, Jérôme Siméon, Mikio Takeuchi, Olivier Tardieu, and Wei Zhang. META: Middleware for events, transactions, and analytics. *IBM R&D*, 60(2–3):15:1–15:10, 2016.

3 Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. GUS, a frame-driven dialog system. *Artificial Intelligence*, 8(2):155–173, 1977.

4 Derek Bridge. Towards conversational recommender systems: A dialogue grammar approach. In *European Conference on Case Based Reasoning (ECCBR) Workshops*, pages 9–22, 2002.

5 Björn Bringert. Rapid development of dialogue systems by grammar compilation. In *Workshop on Discourse and Dialogue (SIGdial)*, pages 223–226, 2007.

6 Herbert H. Clark and Susan E. Brennan. Grounding in communication. *Perspectives on socially shared cognition*, 13:127–149, 1991.

7 Herbert H. Clark and Edward F. Schaefer. Contributing to discourse. *Cognitive Science*, 13(2):259–294, 1989.

8 Matthias Denecke and Alex Waibel. Dialogue strategies guiding users to their communicative goals. In *European Conference on Speech Communication and Technology (EuroSpeech)*, pages 1339–1342, 1997.

9 Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for knowledge representation. *Reasoning Web*, pages 104–124, 2008.

10 Sumit Gulwani and Mark Marron. NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *International Conference on Management of Data (SIGMOD)*, pages 803–814, 2014.

11 Stephen Holland. Talents in the left brain, 2001. Retrieved Jan., 2017. URL: `http://hiddentalents.org/brain/113-left.html`.

12 IBM. Watson Conversation service. Retrieved Jan., 2017. URL: `https://www.ibm.com/watson/developercloud/conversation.html`.

13 Daniel Jurafsky and James H. Martin. *Speech and Language Processing.* Prentice Hall, second edition, 2009.

14 Ron Kaplan. Beyond the GUI: It's time for a conversational user interface. *Wired*, 2013. URL: `https://www.wired.com/2013/03/conversational-user-interface/`.

15 Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. Learning to transform natural to formal languages. In *Conference on Artificial Intelligence (AAAI)*, pages 1062–1068, 2005.

16 Tobias Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, 2014.

17 Chia-Wei Liu, Ryan Lowe, Iulian V. Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation, 2016. URL: `http://arxiv.org/abs/1603.08023v1`.

18 Bruce Lucas. VoiceXML for web-based distributed conversational applications. *Communications of the ACM (CACM)*, 43(9):53–57, 2000.

19 Michael F. McTear. Spoken dialogue technology: Enabling the conversational interface. *ACM Computing Surveys (CSUR)*, 34(1):90–169, 2002.

20 Aarne Ranta. Grammatical Framework: A type-theoretical grammar formalism. *Journal of Functional Programming (JFP)*, 14(2):145–189, 2004.

21 Mandana Vaziri, Olivier Tardieu, Rodric Rabbah, Philippe Suter, and Martin Hirzel. Stream processing with a spreadsheet. In *European Conference on Object-Oriented Programming (ECOOP)*, pages 360–384, 2014.

22 Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. PARADISE: A framework for evaluating spoken dialogue agents. In *Association for Computational Linguistics (ACL)*, pages 271–280, 1997.

23 Jason D. Williams, Nobal B. Niraula, Pradeep Dasigi, Aparna Lakshmiratan, Carlos Garcia Jurado Suarez, Mouni Reddy, and Geoff Zweig. Rapidly scaling dialog systems with interactive learning. In *International Workshop on Spoken Dialog Systems (IWSDS)*, pages 1–13, 2015.