Cogent: A Generic Dialogue System Shell Based on a Collaborative Problem Solving Model

Lucian Galescu, Choh Man Teng, James Allen, Ian Perera
Institute for Human and Machine Cognition (IHMC)
40 S Alcaniz, Pensacola, FL 32502, USA
{lgalescu,cmteng,jallen,iperera}@ihmc.us

Abstract

The bulk of current research in dialogue systems is focused on fairly simple task models, primarily state-based. Progress on developing dialogue systems for more complex tasks has been limited by the lack generic toolkits to build from. In this paper we report on our development from the ground up of a new dialogue model based on collaborative problem solving. We implemented the model in a dialogue system shell (**Cogent**) that allows developers to plug in problem-solving agents to create dialogue systems in new domains. The Cogent shell has now been used by several independent teams of researchers to develop dialogue systems in different domains, with varied lexicons and interaction style, each with their own problem-solving backend. We believe this to be the first practical demonstration of the feasibility of a CPS-based dialogue system shell.

1 Introduction

Many areas of natural language processing have benefited from the existence of tools and frameworks that can be customized to develop specific applications. In the area of dialogue systems, there are few such tools and frameworks and they mostly remain focused on simple tasks that can be encoded in a state-based dialogue model (see, e.g., Williams et al., 2016 and the Dialogue State Tracking Challenge[1]). In this category some of the more expressive approaches to dialogue modeling are based on the information state (Cooper, 1997); notable toolkits include TrindiKit (Larsson and Traum, 2000) and its open-source successor trindikit.py (Ljunglöf, 2009), and OpenDial (Lison and Kennington, 2016).

Unfortunately, there is a dearth of tools for developing mixed-initiative dialogue systems that involve complex back-end reasoning systems. Early theoretical work of SharedPlans (Grosz and Kraus, 1996; Lochbaum et al., 1990) and plan-based dialogue systems (e.g., Allen and Perrault, 1980; Litman and Allen, 1987) laid good foundations. The Collaborative Problem Solving (CPS) model (Allen et al., 2002) seemed to promise a solution but that model has never been implemented in a truly domain-independent way. Ravenclaw (Bohus and Rudnicky, 2009) is a plan-based dialog management framework that has been used to develop a number of dialogue systems. Its dialogue engine is task-independent and includes a number of generic conversational skills; however, its behavior is driven by task-specific dialogue trees, which have to be implemented anew for every application.

Dialogue management involves understanding the intention of the user's contributions to the dialogue, and deciding what to do or say next. It is the core component of a dialogue system, and typically requires significant development effort for every new application domain. We believe that dialogue managers based on models of the collaborative problem solving process offer the highest potential for *flexibility* and *portability*. Flexibility refers to the ability to cover the full range of natural dialogues users may want to engage in, and portability refers to how easy it is to customize or modify a system to work in new domains (Blaylock, 2007).

In this paper we describe a new, domain-independent dialogue manager based on the CPS model, and its implementation in an open-source dialog system shell (**Cogent**[2]). To demonstrate its flexibility, we also describe briefly a few dialogue systems for different domains.

[1] https://www.microsoft.com/en-us/research/event/dialog-state-tracking-challenge/

[2] https://github.com/wdebeaum/cogent

Proceedings of the SIGDIAL 2018 Conference, pages 400–409,
Melbourne, Australia, 12-14 July 2018.

2 Collaborative Problem Solving

When agents are engaged in solving problems together, they need to communicate to agree on what goals to pursue and what steps to take to achieve those goals, negotiate roles, resources, etc. To underscore its collaborative aspect, this type of joint activity has been called Collaborative Problem Solving (CPS). Modeling the type of dialogue agents engage in during CPS must, therefore, take into account the nature of the joint activity itself. In the early 2000s, Allen and colleagues described a preliminary plan-based CPS model of dialogue based on an analysis of an agent's collaborative behavior at various levels:

- An **individual problem-solving** level, where each agent manages its own problem-solving state, plans and executes individual actions, etc.;
- A **collaborative problem-solving** level, which models and manages the joint or collaborative problem-solving state (shared goals, resources, situations);
- An **interaction** level, where individual agents negotiate changes in the joint problem-solving state; and, finally,
- A **communication** level, where speech acts realize the interaction level acts.

This model was refined in a series of publications, and several prototype systems were developed for illustration (Allen et al., 2002; Blaylock and Allen, 2005; Allen et al., 2007; Ferguson and Allen, 2007), all based on the TRIPS system (Allen et al., 2000).

One of the main benefits of this model is that linguistic interpretation and high-level intention recognition could be performed independently of the individual problem-solving level, whose contribution to interpretation would be to specialize the higher-level intentions into concrete problem-solving actions and verify that such actions make sense. The corollary is that in this model the back-end problem solvers would be insulated from the need to worry about linguistic issues.

On this basis, it should be possible to create a generic dialogue system shell with only domain-independent components. Other developers, not necessarily specialists in NLU or dialogue systems, could use this shell to build, relatively quickly, intelligent dialogue systems for collaborative tasks in various domains. The various prototypes of TRIPS CPS-based systems referenced above did not fulfill this promise. In each, the CPS level was integrated fairly tightly with the individual problem-solving level for the application domain, and they were all developed by the same team. Thus, even though each such prototype implemented (a version of) the CPS model and used the same platform for NLU, the ultimate goal of creating a domain-independent dialogue shell that others could customize to develop independently dialogue systems has so far remained elusive. Similarly, the CPS-based dialogue manager in SAMMIE (Becker et al., 2006) also aimed for domain independence but never quite realized it (Blaylock, 2007).

In the rest of the paper we will report on our attempt to develop a generic dialogue shell based on the CPS model. We start with a description of the general architecture of a dialogue system based on the CPS model. Then, we will describe our dialogue manager, with a focus on its interface with the domain-specific problem solving agent. Finally, we give some details on six prototype dialogue systems developed using our dialogue shell, five of which were developed by independent teams of researchers.

3 CPS-based Dialogue Systems

A collaborative conversational agent must understand a user's utterances, that is, obtain a representation of the meaning of the utterance, recognize its intention, and then reason with this intention to decide what to do and/or say next. Finally, the system must convert its own intentions into language and communicate them to the user.

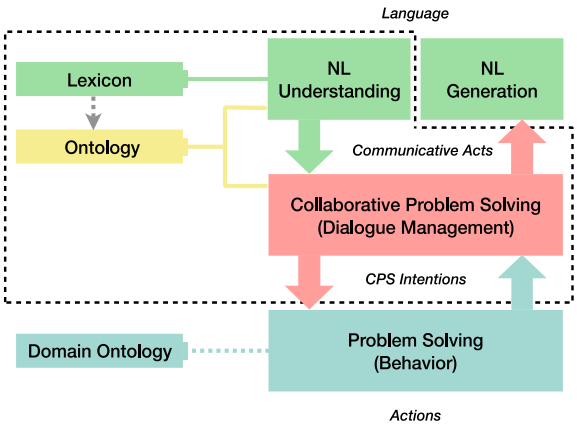

Figure 1: Conceptual architecture of a CPS-based dialogue system

Figure 1 shows a conceptual diagram of the dialogue system we envision. This follows the common separation of a conversational agent's functionality into *interpretation*, *behavior* and *generation*, but where the separation lines are is critical for realizing the idea of isolating domain-independent from domain-specific processing. We take the output of NL Understanding (assumed here to have broad lexical, syntactic and semantic coverage) to be a domain-independent semantic representation of the user's utterance (a *communicative act*), expressed in terms of a domain-independent ontology. Intention recognition is performed by the CPS agent, which takes into account the discourse context and converts communicative acts into *abstract communicative intentions*. These communicative intentions need to be further evaluated with respect to the actual problem-solving state, so they are not fully interpreted until they reach the problem solving agent. This agent is responsible for the domain-specific behavior – hereafter we will refer to it as the *Behavioral Agent* (BA) – and for operationalizing the communicative intentions into actions (which may involve planning, acting on the world, updating its knowledge of the situation, etc.). An autonomous BA should be able to plan and act on its own, but neither the BA nor the user can singlehandedly decide on the status of collaborative goals without a commitment from the other party. The BA expresses its attitude towards shared goals by sending to the CPS agent its own communicative intentions, which the CPS agent will use to update the collaborative state and generate communicative acts for NL generation (such as accepting or rejecting a goal, or proposing a new one).

Customization: Figure 1 includes, on the left side, a number of resources needed by our ideal dialogue system: (1) a broad *lexicon* for NL understanding; (2) a general-purpose (upper-level) *ontology*; and, optionally, (3) a *domain ontology*.

Even a state-of-the-art broad coverage parser, with an extensive domain-independent high-level ontology and lexicon, will not contain all the word senses and concepts needed for every application domain. Additionally, the general ontology concepts need to be mapped onto the domain ontology used by the back-end problem solvers.

Lastly, NL generation from semantic representations of communicative acts is a difficult problem, with no general solutions. Many task-oriented dialogue systems employ template-based techniques, which can lead to satisfactory, if somewhat repetitive text realizations. Such templates are tailored for the application domain.

It may appear that customizing a generic dialogue shell to specific applications involves a considerable amount of work. Nevertheless, we believe these customization tasks are easier to accomplish and require less linguistic expertise than building a dialogue manager for every application, let alone building domain-specific natural language understanding components.

4 Our CPS Model

Let us now turn to the details of our new instantiation of the CPS model. Unlike prior work on CPS-based dialogue management, we focus on the interface between the CPS agent (CPSA) and the BA. This allows us to directly address the issue of domain-independence that posed difficulties in other approaches (e.g., Blaylock, 2007).

The CPSA computes communicative intentions based on the communicative acts resulting from the NLU component. These communicative intentions are realized in our model as *CPS Acts*, represented as a pair *<ACI, CONTEXT>*, where *ACI* represents the abstract communicative intention and *CONTEXT* represents the semantic content of the act in a knowledge representation language. Where there is no ambiguity we will omit *CONTEXT* and denote CPS acts by their *ACI* only.

In the following subsections we will describe the set of CPS acts we have devised so far, grouped by the manner in which they affect the collaborative state.

4.1 CPS Acts Related to Problem-Solving Objectives

The CPS Model defines an *objective* as an intention that is driving the agent's current behavior (Allen et al., 2002). An objective can be proposed by either agent, provided they are ready to commit to it. We represent the intention to commit to an objective via the CPS act ADOPT. For example, if the user starts a conversation with "*Let's build a tower*", this results in the following CPS act:

```
(ADOPT :id O1 :what C1 :as (GOAL))
```

Here, O1 represents a unique, persistent identifier for the shared objective proposed via this act (all objectives are assigned an identifier). C1 is an identifier indexed into the *CONTEXT* of this CPS act (i.e., it refers to an event of building a tower). Additionally, the act also indicates the relation between this objective and any pre-existing objectives. In this example, the relation was identified as GOAL, indicating that this is a top-level objective (we will discuss later other types of relations between objectives available in our model).

Once an objective has been jointly committed to, either agent can propose to drop their commitment to it, via a CPS act called ABANDON. Or, they might propose to shift focus from the active objective (the one currently driving the agents' behavior), by an act called DEFER, which will result in the objective becoming inactive. A proposal to bring an inactive objective back into an active state an agent results in a SELECT act. Finally, an agent can propose that an objective should be considered completed, via a RELEASE act. All these four acts only take as an argument the objective's unique identifier, for example: `(ABANDON :id O1)`.

Note that all of these four acts can be *proposed*, indicating the agent's intentional stance towards their commitment to that objective. The user performs a proposal via a speech act. The same intention may be expressed by different surface speech acts. Going back to our example, the objective of building a tower together can be expressed via a direct proposal ("*Let's build a tower*"); a question ("*Can we build a tower?*"); or an indirect speech act ("*I think we should build a tower*"). The CPSA recognizes the user intent in all these variants, using the surface speech act and other linguistic cues present in the communicative act it receives from NLU). Thus, they all result in the same ADOPT act as above.

If, on the other hand, the BA wants to propose that an objective be jointly pursued, say that it wants to start working on O1 by a subgoal O2 of placing a block on the table, it can do so via a PROPOSE act, whose content is the intention to commit to that objective:

```
(PROPOSE :content (ADOPT :id O2
 :what C2 :as (SUBGOAL :of O1)))
```

where C2 is indexed into the *CONTEXT* of the act for a representation of the event of placing a block on the table. Upon receiving this act, the CPSA will update the collaborative state to reflect the BA's intention to commit to O2, and formulate a communicative act for NLG to realize the proposal in a system utterance.

For a proposal to result in a shared objective, the two agents must agree to commit to it. The CPSA is responsible for gathering the agreements of both the user and the BA. When the CPSA recognizes that the user is proposing an objective, it will first send an EVALUATE act to the BA, whose content is the proposed objective, e.g.,:

```
(EVALUATE :content (ADOPT :id O1
 :what C1 :as (GOAL))
```

This act creates an obligation on the part of the BA to evaluate whether it is able to commit to it in the current situation, and, if so, respond by signaling agreement (ACCEPTABLE), rejection (REJECTED), or, when it cannot even interpret what the objective is, a failure (FAILURE). For example, the BA's agreement, that is, its intention to commit to the objective proposed by the user, would be communicated via:

```
(ACCEPTABLE :content (ADOPT :id O1
 :what C1 :as (GOAL))
```

Since the user has already signaled their intention to commit to the objective by proposing it, on receiving from the BA that the objective is ACCEPTABLE, the CPSA knows that there is mutual agreement, decides that that the objective is now adopted, and sends back to the BA the following CPS act:

```
(COMMIT :content (ADOPT :id O1
 :what C1 :as (GOAL))
```

to signal that now there is a joint commitment to O1. This creates an obligation on the part of the BA to pursue O1 in whatever manner it deems appropriate.

When we have a system-proposed objective, such as O2 above, if the user expresses their acceptance ("*Yes*", "*Sure*", "*I can handle that*", etc.), the CPSA will recognize this as completing the agreement, and then it would adopt the objective and send the COMMIT act to the BA.

Having described in some detail how objectives are created, and how the CPSA decides that there is joint commitment to them, let us turn briefly to some of the details that we brushed over.

Relations between objectives: We mentioned above two relations between the objective currently under consideration and the prior objectives (either previously adopted ones, or ones that have been discussed but are still being negotiated), namely GOAL and SUBGOAL. Currently the CPSA can infer two more. One is MODIFICATION, used when one of the agents is expressing an intention of changing in some manner a prior objective (for example, if one of the agents had suggested placing a blue block on the table, the other agent might suggest placing a red block instead). The second one we call ELABORATION, and is used by the CPSA to signal that it has insufficient knowledge to decide whether the objective under discussion is really a subgoal or a modification of another one, or, perhaps a new top-level goal. It is possible, however, that the BA may be able to use its more detailed knowledge of the situation to make that determination. Thus, upon receiving an objective marked as an elaboration of another one, if the BA deems it acceptable, it has the obligation to clarify the relation as well.

Rejections and failures: If a user proposes an objective, presumably they have an expectation that the objective is achievable. If the BA rejects it, the user will likely not be satisfied with a simple "*No*". Similarly, if the BA fails to understand the objective (or if it encounters any other type of failure, e.g., while trying to perform some action), the system should be able to explain what happened. Thus, the REJECTED and FAILURE CPS acts have features for optionally specifying a reason and a possible way of repairing the situation. The reason for rejection/failure is one of a relatively small set of predefined ones (e.g., UNKNOWN-OBJECT, FAILED-ACTION), and it is expected that the NLG component will make use of it to generate more helpful utterances. As for how to repair the situation, this can be an alternative objective, that the BA is ready to commit to, which could be either a modification of the rejected one, or, perhaps, an objective which, if realized, would make the rejected objective acceptable. For example, if the user wanted to build an all-blue 5-block tower, but the BA has only 4 blue blocks, it would reject the goal (INSUFFICIENT-RESOURCES), but it could suggest as an alternative that a 4-block blue tower would be an achievable alternative. This might be realized as "*Sorry, I don't have enough blocks for that, but we can build a 4-block blue tower.*". If the user accepts ("*OK*"), the CPSA will immediately commit to the suggested objective.

4.2 CPS Acts Related to Situations

Collaborative problem solving requires not only joint commitments to certain objectives, but also a set of shared beliefs about the situation. These shared beliefs occasionally need to be updated. One agent may inform the other of a fact that they believe the other should know. This may come about unprompted or as a result of being asked. The CPS Model offers little guidance on how such acts fit in, even though they are very common in conversation. The examples given seem to suggest an interpretation of questions and simple assertions based on plan recognition (Allen, 1979), which is a tall order, particularly for a domain-independent dialogue manager. When agent A informs agent B of a fact P, this indicates A's immediate intention that B knows P. Similarly, if A asks B whether P is true (an ask-if speech act) or what object satisfies P (an ask-wh speech act), A's immediate intention is that B informs A of those particular facts (Allen and Perrault, 1980). Getting at the intentions *behind* these immediate intentions requires fairly sophisticated, often domain-specific reasoning (in our implementation the CPSA can do that to some extent via abstract task models, but, due to space limitations, we will not discuss it here). Therefore, we created a small set of CPS acts for representing the intentions to impart and request knowledge about situations.

In our model, an assertion of a fact results in the following CPS act:

```
(ASSERTION :id A3 :what C3
 :as (CONTRIBUTES-TO :goal O1))
```

where C3 is an identifier pointing to a representation of the content of the assertion in the *CONTEXT* of the CPS act. The relation between an ASSERTION act and an existing objective (or NIL if no such objective exists) is an underspeci-

fied one, of contributing somehow to it. The BA needs to decide, if it accepts A3, how this addition will change its understanding of the situation and affect O1 or any other (adopted) objective.

For ask-if questions the CPSA will produce the following act:

```
(ASK-IF :id A4 :query Q4
 :as (QUERY-IN-CONTEXT :goal O1))
```

Here Q4 is an identifier pointing to a representation (in the *CONTEXT* of the CPS act) of a statement to be evaluated for its truth value.

For ask-wh questions the CPSA produces acts in the following format:

```
(ASK-WH :id A5
  :what W5 :query Q5 :choices S5
  :as (QUERY-IN-CONTEXT :goal O1))
```

This expresses the intention of knowing the value of an entity (W5), possibly restricted to a set of choices (S5), that makes a proposition (Q5) true. As before, all these identifiers should be given appropriate descriptions in the *CONTEXT*. This act can thus represent the intention expressed by a question such as "*What color should we use for the first block, blue or red?*".

Finally, an answer to a question takes the following form:

```
(ANSWER :to A5
 :what W5 :query Q5 :value V6
 :justification J6)
```

This indicates the value V6 (e.g., blue) for the entity W5 makes the statement Q5 true (we should use blue for the first block), in response to the CPS act with the identifier A5. If the answer is in response to an ASK-IF act, V6 can only be TRUE or FALSE. Optionally, a justification (J6) may be added to show how the answer came about.

It is important to note that we treat these intentions as special types of objectives, that can become adopted, active, etc., just like other objectives. For example, if one of these CPS acts is initiated by the user, the act must be evaluated by the BA. If it deems the act ACCEPTABLE, the CPSA will commit to working on it (updating the system's beliefs, or answering the question). If originating from the BA, the act must be proposed first, and realized through a communicative act.

Side effects: We noted above that updating the system's beliefs about the situation may affect the status of existing objectives. Insofar as the BA is capable of foreseeing these effects, it ought to inform the CPSA so the collaborative state can be updated. Any such changes would result in an obligation to inform the user. In our model we use an additional feature for the ACCEPTABLE act (see previous section), for describing the effect. Its value is an objective to be proposed. For example, if, in the context of the shared objective of building a tower, the system asks "*Who is going to move the blocks?*", and the user says "*I will*", this answer has the side effect of modifying the existing objective (in this case specializing it to include the identity of the builder). The system's acceptance of the answer will necessarily imply the acceptance of the modification as well, and the CPSA will update the collaborative state accordingly.

4.3 CPS Acts Related to Initiative and Execution

Another important role of the CPSA in managing the dialogue is to negotiate initiative. To facilitate an orderly conversation, it restricts both the timing and the magnitude of the BA's ability to affect the collaborative state. It does so via a special CPS act, called WHAT-NEXT, which takes a single argument: the identifier of an adopted shared objective (usually the one that is active). This act can be sent to the BA whenever there are no pending updates to the collaborative state, and no outstanding communicative acts to process or to wait on. In effect, by sending this act, the CPSA transfers the task initiative to the BA, which gives it the chance to, ultimately, influence discourse initiative as well. The BA has the obligation to respond with a single update to the collaborative state, presumably the one with the highest priority. This restriction is critical, because it frees the CPSA from the need to consider too many options about what to do and say next, a decision that, in many situations, would require domain-specific knowledge.

The BA's reply to a WHAT-NEXT depends on its own private problem-solving state. It may be that it has done some planning and, as a result, it wants to propose a way of making progress towards accomplishing the active objective. It may be that it does not have sufficient information to make progress, in which case it may formulate an intention to ask the user to provide the information. Or, if the active objective is a question, it may have come up with an answer; that update would prob-

ably get very high priority. All these possibilities are handled by acts we have already discussed.

One other possibility is that the BA is currently not doing any reasoning, but simply acting on the active objective, or has accomplished it. Updates to the status of an objective are communicated via a special CPS act, which takes the following form:

```
(EXECUTION-STATUS :goal A1
 :status GS)
```

Here `GS` is an expression that indicates the status of the goal. Currently it can be one of three indicators:

1. `DONE`, which signifies that `A1` was accomplished. CPSA will create a communicative act to inform the user, and, if the user agrees, releases the objective.

2. `WORKING-ON-IT`, which indicates that the BA is actively pursuing `A1`, but it will take more time. The CPSA may decide to inform the user, and creates a trigger for itself to check back later.

3. `WAITING-FOR-USER`, which indicates that the BA cannot make progress on `A1` because it is waiting for the user to act on it (or another objective that `A1` depends on). As a result, the CPSA will construct a communicative act to prompt the user.

This CPS act also allows the BA to communicate partial execution status (that it has executed some actions, though it has not accomplished the objective yet), but we leave those details out of this discussion.

5 The Cogent System

We implemented our CPS model as a component in the TRIPS system (Allen et al., 2000), which has recently been released in the public domain under a GNU GPL License.

The TRIPS system comes with a broad coverage parser (Allen and Teng, 2017) with an extensive grammar and an effective 100,000+ word semantic vocabulary defined in terms of a 4000 concept domain-independent ontology. It operates in concert with a suite of statistical preprocessing components, performing tasks such as part-of-speech tagging, named entity recognition, and identification of likely constituent boundaries. These preprocessed inputs are provided to the core TRIPS parser as advice. The parser constructs from the input a logical form, which is a semantic representation that captures an unscoped modal logic (Manshadi et al., 2008). The logical form includes the surface speech act, semantic types, semantic roles for predicate arguments, and dependency relations.

TRIPS also includes an interpretation manager that converts the logical forms into communicative acts, performing language-based intention recognition and normalizing different surface forms.

We packaged the TRIPS NLU components (including the lexicon and ontology) with our CPS agent, thereby creating a dialogue system shell, which we call **Cogent**. This system does not include a BA or an NLG component (**Cogent**'s components are surrounded with a dashed line in Figure 1). Thus, it is a true domain-independent shell, not a system that can be adapted to other domains. It can carry out very minimal conversations because social conversational acts such as greetings are handled in a domain-independent manner in the CPSA. But, ultimately, the purpose of the shell is to be used to create domain applications. The success of the task we set to accomplish is whether this shell can be and is used by independent developers to develop operational dialogue systems in domains of their choice.

As discussed in the previous section, the CPS acts and the obligations they engender establish a protocol that developers of behavioral agents must implement. Other than that, we believe the CPSA offers functionality to develop different styles of conversational agents (user-driven, system-driven or fully mixed-initiative). The developers also must implement their own NL Generation component, for reasons that we touched upon earlier.

Of note, by default all CPS acts have their contents expressed in the TRIPS ontology. We are also providing a tool for mapping concepts in the TRIPS ontology to domain ontologies. We have adapted the TRIPS interpretation manager to use these mappings to produce content in the domain ontology, to make it easier for the Behavioral Agents to interpret the *CONTEXT* associated with each CPS act. The details of the ontology mapping tool and the mappings it creates are, however, beyond the scope of this paper.

6 Systems Implemented in Cogent

We describe briefly six system prototypes that have been built using **Cogent** as the base frame-

work; thus, they all use the same CPS agent described above. In all cases, the developers of these prototypes used the protocol described above to create behavioral agents that, in turn, act as integrators of other problem solvers. The descriptions of these systems are going to be necessarily brief; the interested reader is encouraged to follow the references to get a better understanding of their capabilities and the kinds of dialogues they support (unfortunately, not all systems have been published yet). All these systems have been developed as part of DARPA's Communicating with Computers (CwC) program[3].

Cabot: This is a mixed-initiative system for planning and execution in the blocks world, the tasks being of jointly building structures (Perera et al., 2017). Both the user and the system can come up with their own goals, and, if necessary, they will negotiate constraints on those structures (size, colors, etc.) so all the goals can be completed. They also negotiate their roles in building these structures ("architect" or "builder"). This system uses a 2D simulated version of the blocks world. The examples used in this paper are from interactions with this system.

Cabot-L: This system learns names and structural properties of complex objects in a physically situated blocks world scenario (Perera et al., 2017; Perera et al., 2018). The user teaches the system by providing examples of structures together with descriptions in language. The system has capabilities to perceive the world and detect changes to it, and can ask the user questions about various features of the structures, to learn a general model. To validate the inferred model, the user can then show additional examples and ask the system to classify them and explain its reasoning. The user and the system can interact via either written or spoken language.

BoB: This system acts as an assistant biologist. It has fairly extensive knowledge of molecular biology and can assist the user by responding to inquiries about properties of genes, proteins, molecular mechanisms, their relationship to cellular processes and disease, building and visualizing complex causal models, running simulations on these models to detect their dynamic properties, etc. To manage this wide range of problem-solving behaviors, *BoB*'s BA integrates a variety of agents with specific expertise.

Musica: This system uses a computational model of music cognition, as well as knowledge about existing pieces of music, to help a human composer create and edit a musical score (Quick and Morrison, 2017).

SMILEE: This system acts as a partner for playing a cooperative game (Kim et al., 2018). The game involves placing pieces (blocks) on a board to create complex symmetrical configurations. Players alternate, but each player can hold their turn for multiple rounds. Each player has some freedom to be creative with respect to the configuration being pursued (it is not set in advance). Thus, they have to negotiate turn taking, and they can ask for explanations to achieve a shared understanding about the properties of the configuration being created.

Aesop: A system for building animated stories. The user acts as a director, and can choose scenes, props, characters, direct them what to do, etc. Essentially, the system provides a dialogue interface to a sophisticated system for creating visual narratives.

Of note, these systems work in several application domains, with varying interaction styles. *Musica* and *Aesop* currently work mostly in fixed-initiative mode (user tells the system what to do). All others involve varying degrees of mixed initiative. While *Cabot* is a more traditional planning domain, it is interesting to note that all others involve fairly open-ended collaborative tasks, for which the ultimate goal is learning or creating something new. *BoB* is notable for the fact that it is helping the user learn new knowledge, by helping to formulate and evaluate biological hypotheses (which may even lead to new scientific discoveries).

Importantly, with the exception of *Cabot-L*, which was developed by our team, all others were developed by independent teams (the BAs for *Cabot* and *BoB* were developed by a single team, though the latter also involved collaboration with a large group of biologists and bioinformaticians). We helped those teams understand how our tools work and the meaning of the CPS acts (especially to the early adopters, who did not have the benefit of much documentation), but we had no role in deciding what problem-solving behaviors they should or should not implement, how to implement them and so on. Two of the systems (*BoB* and *Musica*) required additions to our surface NLP components (mainly add-

[3] https://www.darpa.mil/program/communicating-with-computers

ing domain-specific named entity recognizers) and some additional ontology concepts and mappings; we provided those customizations. The version of the TRIPS Parser we started with proved to be fairly robust, but we did have to adapt it in response to failures reported by the dialogue system developers. Nevertheless, these enhancements were not domain-specific – that is, the same parser, with the same grammar, is used for all systems.

In all systems, developers used custom template-based NLG.

7 Summary and Discussion

In this paper we reported on the development of a new domain-independent dialogue manager based on the collaborative problem solving model. We packaged this dialogue manager with a suite of broad coverage natural language understanding components (from the TRIPS system) and created a new, domain-independent CPS-based dialogue system shell. This shell has been used by several independent teams of researchers to develop dialogue systems in a variety of application domains, with different conversational styles. We believe this to be the first successful implementation of a domain-independent dialogue system shell based on the CPS model (or any other model of equivalent complexity).

We do not claim the CPSA to be complete, however. For example, it can sometimes detect an ambiguity in the user's intention and generate a clarification question, but its abilities in this regard are fairly limited. *BoB* has demonstrated some limited handling of hypotheticals (in what-if questions) at the problem-solving level, but the CPSA itself does not yet track hypothetical situations. We expect that, with wider adoption, we will inevitably be confronted with the need to improve both our model and its implementation.

As noted above in reference to *BoB* and *Musica*, for domains requiring adaptation of the NLU components, language specialists are still needed. We have not yet endeavored to create tools that would make it easier for dialogue system developers to adapt/improve themselves the NLU components.

Our current focus is on evaluating the robustness of the intention recognition functionality of the CPSA.

Acknowledgments

This research was supported by the DARPA Communicating with Computers program, under ARO contract W911NF-15-1-0542.

References

James F. Allen. 1979. *A Plan-Based Approach to Speech Act Recognition*. Ph.D. Thesis. University of Toronto.

J. Allen, N. Blaylock, and G. Ferguson. 2002. A problem solving model for collaborative agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, pp. 774-781. ACM.

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. 2000. An architecture for a generic dialogue shell. *Natural Language Engineering*, *6*(3-4): 213-228.

J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom, W. 2007. PLOW: a collaborative task learning agent. In *Proceedings of the 22nd National Conference on Artificial intelligence*, Vol. 2, pp. 1514-1519. AAAI Press.

J.F. Allen and C.R. Perrault. 1980. Analyzing intention in utterances. *Artificial intelligence* 15(3):143-178.

J.F. Allen and C.M. Teng. 2017. Broad coverage, domain-generic deep semantic parsing. In *Proceedings of the AAAI Spring Symposium on Computational Construction Grammar and Natural Language Understanding*.

T. Becker, N. Blaylock, C. Gerstenberger, I. Kruijff-Korbayová, A. Korthauer, M. Pinkal, M. Pitz, P. Poller, and J. Schehl. 2006. Natural and intuitive multimodal dialogue for in-car applications: The SAMMIE system. *Frontiers in Artificial Intelligence and Applications*, 141:612.

Nate Blaylock. 2007. Towards Flexible, Domain-Independent Dialogue Management using Collaborative Problem Solving. In *Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue* (*Decalog 2007*), pp. 91–98.

N. Blaylock and J. Allen. 2005. A collaborative problem-solving model of dialogue. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pp. 200–211, Lisbon.

N. Blaylock, J. Allen, and G. Ferguson. 2003. Managing communicative intentions with collaborative problem solving. In *Current and New Directions in Discourse and Dialogue*, pp. 63-84. Springer, Dordrecht.

D. Bohus and A.I. Rudnicky. 2009. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3), 332-361. https://doi.org/10.1016/j.csl.2008.10.001

Robin Cooper. 1997. Information states, attitudes and dialogue. In *Proceedings of the Second Tbilisi Symposium on Language, Logic and Computation*, Tbilisi, pp. 15-20.

G. Ferguson and J. Allen. 2007. Mixed-initiative systems for collaborative problem solving. *AI magazine*, *28*(2):23.

B.J. Grosz and S. Kraus. 1996. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269-357.

S. Kim, D. Salter, L. DeLuccia, K. Son, M.R. Amer, and A. Tamrakar. 2018. SMILEE: Symmetric Multi-modal Interactions with Language-gesture Enabled (AI) Embodiment. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2018)*.

S. Larsson and D.R. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 6(3-4), 323-340.

P. Lison and C. Kennington. 2016. OpenDial: A toolkit for developing spoken dialogue systems with probabilistic rules. In *Proceedings of ACL-2016 System Demonstrations*, pp 67-72. Association for Computational Linguistics. https://doi.org/10.18653/v1/P16-4012

D.J. Litman and J.F. Allen. 1987. A Plan Recognition Model for Subdialogues in Conversations. *Cognitive Science*, 11: 163-200. https://doi.org/10.1207/s15516709cog1102_4

Peter Ljunglöf. 2009. trindikit.py: An open-source Python library for developing ISU-based dialogue systems. In *Proceedings of the 1st International Workshop on Spoken Dialogue Systems Technology (IWSDS'09)*, Kloster Irsee, Germany.

K.E. Lochbaum, B.J. Grosz, and C.L. Sidner. 1990. Models of plans to support communication: An initial report. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 485-490.

M.H. Manshadi, J. Allen, and M. Swift. 2008. Toward a universal underspecified semantic representation. In *Proceedings of the 13th Conference on Formal Grammar (FG 2008)*, Hamburg, Germany.

V. Pallotta. 2003. Computational dialogue models. MDM research project deliverable, EPFL IC-ISIM LITH, Lausanne (CH).

I.E. Perera, J.F. Allen, L. Galescu, C.M. Teng, M.H. Burstein, S.E. Friedman, D.D. McDonald, and J.M. Rye. 2017. Natural Language Dialogue for Building and Learning Models and Structures. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, pp. 5103-5104.

I. Perera, J. Allen, C.M. Teng, and L. Galescu. 2018. A Situated Dialogue System for Learning Structural Concepts in Blocks World. In *Proceedings of the 19th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL 2018)*, Melbourne, Australia.

D. Quick and C.T. Morrison. 2017. Composition by Conversation. In *Proceedings of the 43rd International Computer Music Conference*, pp. 52-57.

J. Williams, A. Raux, and M. Henderson. 2016. The dialog state tracking challenge series: A review. *Dialogue & Discourse*, 7(3), 4-33.