Approaches for Dialog Management in Conversational Agents

Harms, Jan-Gerrit; Kucherbaev, Pavel; Bozzon, Alessandro; Houben, Geert-Jan

DOI
10.1109/MIC.2018.2881519

Publication date
2019

Document Version
Final published version

Published in
IEEE Internet Computing

Theme Article

Approaches for Dialog Management in Conversational Agents

Jan-Gerrit Harms
Delft University of Technology

Alessandro Bozzon
Delft University of Technology

Pavel Kucherbaev
Delft University of Technology

Geert-Jan Houben
Delft University of Technology

***Abstract*—Dialog agents, like digital assistants and automated chat interfaces (e.g., chatbots), are becoming more and more popular as users adapt to conversing with their devices as they do with humans. In this paper, we present approaches and available tools for dialog management (DM), a component of dialog agents that handles dialog context and decides the next action for the agent to take. In this paper, we establish an overview of the field of DM, compare approaches and state-of-the-art tools in industry and research work on a set of dimensions, and identify directions for further research work.**

The dream of a human-like highly intelligent computer assistant has been presented in many science fiction movies like *Hal* in "2001: A Space Odyssey" (1968), *Samantha* in "Her" (2013), and *Jarvis* in "Iron Man" (2013). Recent advances in automatic speech recognition systems, machine learning, and artificial intelligence enabled the advent of personal assistants like *Google Assistant*, *Siri,* and *Alexa*, first on smartphones and lately on home speakers and other devices. These might make the reality seem very close to the fiction in the movies. However, while the assistants are capable of executing small tasks, the richness and quality of their dialogs are not comparable to the ones of humans: interactions are still simple, short, and constrained by a limited vocabulary, thus forcing users to adjust to the system's capabilities.

Digital assistants are part of a larger group of dialog agents which include: *voice user interfaces* (or spoken dialog systems), *text-based agents* and *embodied conversational agents*,[1] [Chapter 4]. Historically, dialog agents aimed to simulate human conversation. The first examples of text-based agents are ELIZA[2] which acted as a Rogerian psychotherapist, and PARRY[3] simulating a paranoid schizophrenic. This was possible with extensive rule-sets and structured question-answer sets. With advances in *natural language processing*

Digital Object Identifier 10.1109/MIC.2018.2881519
Date of publication 15 November 2018; date of current version 8 April 2019.

Published by the IEEE Computer Society

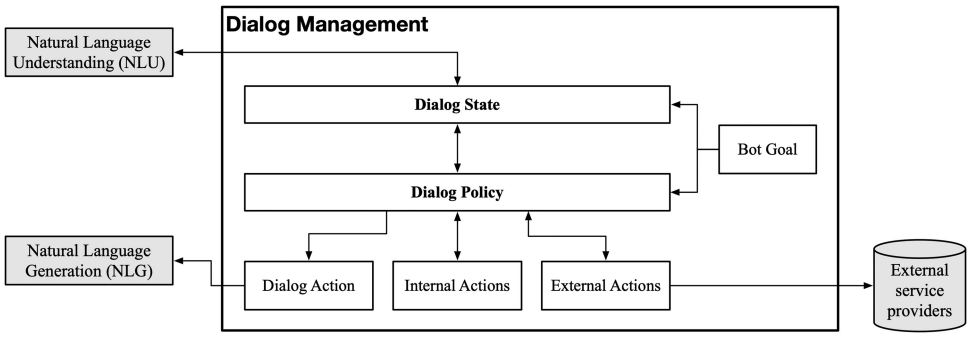

Figure 1. DM system architecture and information flow.

it became possible to have goal-oriented conversations by extracting pieces of information from user utterances and using that information for digital requests to external services. This type of dialog is still prevalent and is the underlying concept of most commercial systems,[4] [Chapter 28]. But dialogs with digital assistants are short and simple, mostly not more than two interactions, while many tasks like travel planning can contain many more steps,[4] [Chapter 29]. Creating rule sets for such complex dialogs is cumbersome and results in brittle dialog systems which have problems handling unforeseen user input. That is why research work is looking into using probabilistic techniques to learn a *dialog policy*, i.e., the strategy of what to say next in the conversation, from transcriptions of real conversations. Among these techniques, systems based on partially observable Markov decision processes (POMDP) received great attention.[5] POMDPs model the conversation state as a not fully observable statistical variable. Lately, researchers shifted their focus to *neural network-based approaches*[6] and *hybrid approaches* emerged that use a combination of rule-based and probabilistic dialog policies.[7] Still, despite many years of research work, both the scientific and industrial world struggle to understand which approaches and tools are optimal for the type of dialog agent they plan to create.

As for now, dialog agents are still a convenience tool performing mini-transactions and need major innovations in order to become as useful as their fictional counterparts. In this paper, we discuss the approaches and tools for implementing the dialog management (DM) component of a conversational agent. The dialog manager keeps track of the information exchanged in the dialog and decides upon the next action of the dialog agent. Specifically, the purpose of this paper is three-fold:

1. to explore the ways in which DM has been approached;
2. to provide an overview of the state-of-the-art of commercial as well as research tools;
3. to define opportunities for future research directions.

First, we introduce the main concepts and terminology pertaining to the DM in conversational agents. Then, we introduce seven evaluation dimensions used to assess and compare the properties of seven groups of tools. The comparison provides an overview of the state-of-the-art and enables a discussion on directions for future research work.

DIALOG MANAGEMENT

Figure 1 shows the system architecture and information flow of a DM system. Dialog agents receive requests from users either through spoken language or direct text input and outputs either a textual or vocal (through speech synthesis) response. The shown architecture is applicable to both speech and text, although in the former case, spoken words need to be converted to text before the natural language understanding (NLU).

The figure is best explained by giving an example. Assume our dialog system is a travel planning service and the user tries to book a flight for a business trip. Once a message is received (e.g., "Book a flight to Amsterdam") it is first handled by the NLU unit that converts the

message to a user action, also called the user intent (e.g., intent: "FlightBooking").

This NLU's output can carry data fields, also called slots (e.g., location: "Amsterdam"), which the user tries to convey to the dialog agent. The dialog manager then uses this output to update the state of the conversation. The DM and NLU are separate components, but they influence each other's performance. For instance, a powerful NLU can also make the DM more capable and shorten the conversation (e.g., no need for extra confirmation messages, if the system is confident in understanding the user). If some slot (e.g., destination city) is not extracted the dialog manager might need to request destination while if the slot is filled it can directly ask for the next missing slot (e.g., departure city). On the other hand, inputs from DM can help the NLU in adapting its internal operations. For instance, knowledge about the next required input type (e.g., a destination city) could enable the NLU to employ domain- and entity-specific models (e.g., one trained to recognize city name). The *dialog state* keeps track of any information received throughout the conversation. It forms the foundation for deciding on the next action and for interpreting the conversation. The dialog state can also be influenced by the *goal* of the dialog agent itself, i.e., objectives pursued by the dialog agent that transcend the immediate intent of the user. For instance, next to helping the user achieve some goal, the agent might also be designed to steer the user behavior into a certain direction. In the travel planner example, the goal of the dialog agent could be to sell a business class flight, so it could propose "There are cheap business class flights available for 1500 Euro" or it could be more subtle hint like "Shall I book it right away?" The same happens when a sales person tries to convince the customer to buy a product.

Once the dialog state is updated, the *dialog policy* is triggered which takes the new state and decides on the next action for the dialog agent to take. The dialog policy is the central piece of the dialog manager, building the bridge between the conversation context, third-party services, and the dialog agent's response. In the context of the dialog policy, some other concepts require an introduction. First, *grounding* is the establishment of common ground between parties in the conversation. This happens through acknowledgment of the last heard user input, through implicit or explicit confirmation. Second, the *initiative* is an important concept; a dialog system can have either user-directed initiative (user is leading the conversation for instance through asking questions to the system), system-directed initiative (the system leads the conversation by requesting information from the user) or mixed initiative (both user and system can take the lead). Finally, *domains* are the fields in which certain actions, states, and intents are defined, in this example the travel or flight domain. Domains can be organized into hierarchies or even graphs of subdomains with each domain having its own policy.[4] [Chapter 28].

The dialog policies choose from *dialog*, *internal* and *external* actions. A dialog action corresponds to a message output that is sent to the user which can either be a template "There is a flight at {departure_time}" or, in more complex systems, a dialog act, for instance, to inform the user (inform(flight = "AE23", departure_time = "1 pm")). This output will then be converted by the natural language generation (NLG) component to a textual response to the user. An internal action is one that the dialog agent orchestrates in order to modify its behavior or improve its performance: for example, improving the policy through retraining, or seeking external input for performance improvement (e.g., in hybrid conversational agents,[8] or in systems with online learning capabilities). An external action interacts with a service provider to satisfy a user's request, by requesting data or by triggering some application event. In our example that would correspond to getting a list of flights and submitting a booking request once all information is available. Also, multiple actions can be possible: when the request for the flight booking service is taking some time the dialog agent might inform the user "I am looking for available flights right now."

DIMENSIONS OF ANALYSIS

In this paper, we assess DM approaches and tools with respect to the following aspects:

- capability of creating natural, robust and complex dialogs;

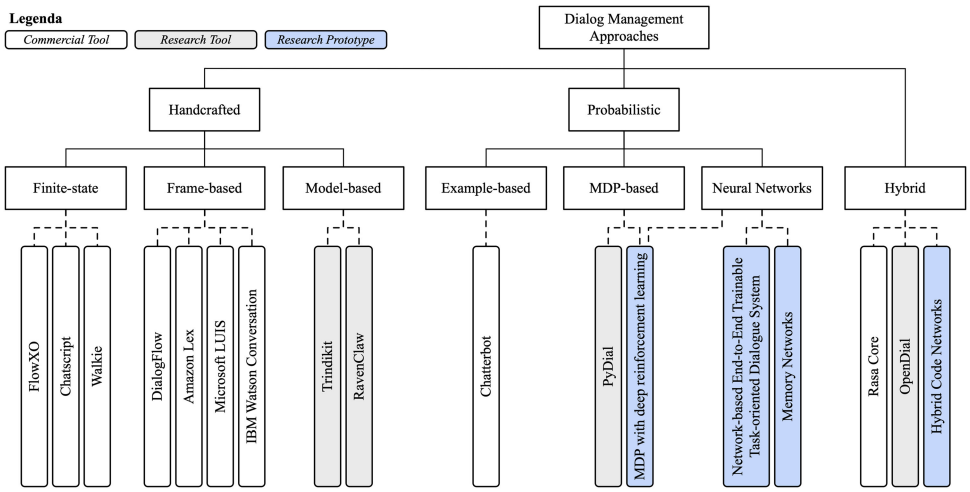

Figure 2. Taxonomy of DM approaches. On the top-level three different approaches can be identified, namely handcrafted systems, statistical systems, and hybrid systems. Below the subcategories, available commercial tools (white boxes), research tools (grey boxes), and research prototypes (blue boxes) are present.

- convenience for developers;
- applicability in a commercial environment;
- scalability/reusability in multiple applications.

Towards this goal, we derive the following dimensions of analysis.

Dialog structure. Dialogs can often be simplified by modeling it as a structure of possible states and state-transitions. A dialog structure could be a linear sequence of messages or a tree-like structure.

Learning. With the growing complexity of dialog agents, it will be important that the dialog strategies improve automatically when more data becomes available through new interactions. This setting tells whether improvement of the dialog manager is possible at runtime.

Error handling. Interactions that perfectly fit the conversation the developer had in mind are often easy to handle for any system. Difficulties arise when handling unexpected input and speech recognition or typing errors. This dimension shows how the system can react in such situations.

Dependencies. The dependencies of the tool tell what resources are required to create a working dialog manager. For example, in order to train a model, a corpus of data is required.

Control. For some tasks we require the dialog agent to be very precise, for example when handling the passport identifier in the flight-booking example. In those situations, the developer needs a lot of control about how the dialog agent shall interpret input and react to it.

Domain independence. In order to be reusable in multiple situations, a dialog manager can have some domain independent components, which are unrelated to the topic of the dialog.

Tool availability. For nonexpert developers to use the tool it needs to be convenient to create dialog agents. The availability of the tool tells how developers can access the functionality of the DM software.

APPROACHES AND TOOLS

Figure 2 shows a taxonomy of the approaches for managing dialogs and a classification of a selection of tools. With our selection we do not aim to be exhaustive, taking into consideration the format of this paper. We do aim to provide representative examples of each approach from the taxonomy. For the comparison, we select one tool per approach which helps determine the common features that different implementations based on that approach have. Our taxonomy is based on the one sketched in,[1] where approaches are divided into *handcrafted* (rule-based) and *probabilistic* (statistical). However, some of the tools that were found did fit both categories, which we separated into a third category, *hybrid* systems.

Handcrafted approach. Handcrafted dialog managers define the state of the system as well as the policy by a set of rules which are defined by developers and domain experts. The simplest subset of dialog systems is modeled by a *finite-state automata*, in which the conversation always is in one definite state of the conversation at a time, each state having a fixed number of transitions to other states.

Such dialogs have system-directed initiative, so the system asks information from the user step-by-step.[1] Many equivalent tools exist; for the comparison, we chose Flow XO, a software for business–customer relations such as customer support. It is a hosted solution with a visual interface and many third-party tool integrations.[9]

To offer more flexibility, a data model can be added to the finite-state automaton which keeps track of slots. This type of approach is called *frame-based* DM. Slots are allowed to be filled in any sequence and multiple slots can be filled per turn, thus enabling some user initiative for the semi-mixed initiative system.[4] [Chapter 28]. This is the basis for most commercial task-fulfilling dialog agents. We use Google's DialogFlow (previously api.ai)[10] as an example which is in line with many other similar systems like Amazon Lex and IBM Watson Conversation. Finally, slightly more sophisticated handcrafted dialog systems can have a user model, conversation model or some other model next to the data model. One example is the information state model, which maintains user goal and beliefs in the state and requires the developer to write rules and strategies for updating that information based on dialog acts. As far as we are aware such an approach has not been used in any commercial tools but TrindiKit[11]—a research tool that explored this concept.

Probabilistic approach. Instead of defining rules for the dialog strategy by hand, probabilistic DM takes a different approach by learning the rules from actual conversations. *Example-based* systems learn appropriate answers from a large corpus by matching the last query with an example in the training dataset and uses the response from the training set. For example, a training corpus might contain the conversation set [user: "Hello;" system: "Hi, how are you doing?"]. If the user then starts a conversation with "Hello" a word similar to this, the system will reply with the other message. This was the earliest approach at a statistical dialog manager. It is often used for chatbots that aim to carry open-ended conversations—Chatterbot[12] is an example for an open-source system which uses such an approach—but it suffers from several limitations, especially in terms of error handling. Approaches which have seen a lot of research work more than the past 10 years are *Markov decision processes* (MDP) and especially POMDP. They model the dialog state as an unobserved variable, the belief state, which is a distribution over all possible states, including error ones (e.g., incorrect input from the users). Observations (e.g., user input to the system) provide evidence for the most probable state of the system. Finally, a dialog policy is learned with reinforcement learning to map the belief state to a dialog agent action.[5] In the flight booking example, a message "A flight to Amsterdam," will lead to a high probability that the destination is filled with Amsterdam, while the state with all fields empty is given a low probability. PyDial[13] is a statistical spoken dialogue system toolkit that has been published recently. PyDial features a modular architecture that allows, among others, the adoption of deep reinforcement learning techniques. Even more recently, memory neural networks have been applied to DM. Such neural networks are extended with the ability to read and write to a memory component in order to store information from previous input to the network. They take the pure text as input and return text as well. The goal of this is to use end-to-end learning on a set of dialogs to train DM without any handcrafting of state and action spaces. At the time of writing we were not aware of any tools, so instead, we use a proposed architecture from the paper by Bordes *et al*.[6]

Hybrid approach. Next, to the purely rule or data-based approaches, some work has been done on combining the advantages of both approaches. Such hybrid approaches are an important step toward introducing data-driven elements into available dialog agents. At the time of writing two such tools have been published. *Rasa Core*[14] is a commercial open-source

Table 1. Comparison of DM approaches.

High-level category	Low-level category	Tools	Comparison (dimensions)						
			Dialog structure	Learning	Error handling	Dependencies	Control	Domain independence	Tool availability
Handcrafted	Finite-state	Flow.xo	Tree	manual iterative	escalation message	none	complete control	none	Hosted, Visual
	Frame-based	DialogFlow	Directed acyclic graph	manual iterative	escalation message	none	highly controlled	sub-modules reusable	Hosted, Visual
	Model-based	TrindiKit (Information State)	Undirected graph	manual iterative	rule-based recovery	conversation model	highly controlled	model party reusable	Prolog/Python
Probabilistic	Example-based	Chatterbot	random	no learning	ignored	Large corpus	uncontrolled	examples reusable	Python
	MDP-based	PyDial	Undirected graph	runtime, iteratively	probabilistic recovery	large conversation collection (>1000)	semi-controlled	independent state tracker and policy	Python
	Neural Networks	Memory Networks	Undirected graph	runtime	-	~500 conversation collection	uncontrolled	-	-
Hybrid		OpenDial	Undirected graph	runtime	probabilistic recovery	conversation collection (>100)	semi-controlled	independent state tracker and policy	Java
		Rasa Core	Directed acyclic graph	manual runtime (through interaction)	probabilistic recovery	conversation examples (>10)	semi-controlled	sub-modules reusable	Python

tool that combines frame-based state updates with a learning by example dialog policy. The software implementation is close to the concept of *Hybrid Code Networks*.[15] As the name suggests this approach utilizes neural networks combined with coded constraints and rules. The intuition is that some parts of goal-oriented dialogs, like sorting the data returned by external service providers, is very hard to learn by example dialogs, while it is much simpler to implement in a few lines of code. By implementing software components into the framework, the training data can significantly be reduced. *OpenDial*[7] uses an approach called probabilistic rules, a way to express domain knowledge with *if-then-else* type rules to reduce the overall state space of the underlying POMDP model. For example, if the user is booking a flight to New York and the dialog agent asks the user about the city of departure we can assume with high probability that the user will answer with a city name which is the departure. If we can encode this in rules, this specific part does not need to be learned during the training phase.

COMPARISON

In Table 1, we applied the dimensions to each of the approaches defined in the previous section, taking one representative example tool per DM approach.

The *dialog structure* influences the complexity of the possible dialogs and the naturalness because rigid dialogs are also repetitive. The finite-state approach offers the stiffest structure by modeling dialogs like a tree, so branches in the conversation exist based on the previous answer but the conversation follows a strict flow without returning to previous states. This offers easy engineering of conversations, but at a cost of a limited diversity and richness in dialogs. The tools that use a frame-based dialog state model the dialog structure as directed acyclic graphs because the sequence in which information is given and requested is not predefined. Cycles are usually not possible; once a field is filled, it stays filled. However, correction intent could be manually added, thus making cycles possible. Undirected graphs which are possible with the information state architecture, the neural network and the approaches that use a belief state. In this model, states are activated based on the evidence collected through the whole history or a combination of rules. The flow through the conversation is not defined which makes the conversation less predictable, but arguably more natural. Finally, we have the example driven Chatterbot which has

a semi-random state activation through matching the best example in the corpus. This can result in unexpected answers from the bot making the flow unnatural, but at the same time, if the examples are a good match, agent responses can be surprisingly human-like. For instance, the developer usually thinks about greeting messages that come very naturally however if we ask, "what is the color of tree leaves?," but the only corpus example with the word "color" is "what is your favorite color?," the bot might answer "I like red." which is not an expected answer to the question.

As for *availability* the finite-state tool Flow XO and frame-based DialogFlow, are hosted solutions with visual interfaces for defining messages and frames. This makes them convenient for developers and visual interfaces help nonexpert and even nondevelopers to design interaction with the chatbots. The hosted solutions also offer easy *integration* with external tools that further increases the usefulness for application designers. For instance, in a customer survey application, data requested from the users can directly be fed into a spreadsheet for further processing and analytics. All other tools require writing code to create a dialog agent. TrindiKit was originally written in Prolog, a very specific logic programming language making it very much focused at research. Later a Python implementation became available but developing a working dialog agent still is a daunting endeavor due to the complexity of the underlying rule structure. PyDial, Chatterbot, and Rasa Core all offer Python implementations. Python is a popular language in research work and industry alike, making Python tools targeted at a large group of developers, but nondevelopers are mostly excluded from these tools. Integrations then need to be programmed customized to the application. OpenDial is written in Java, which is a little less common among developers of dialog agents.

It is hard to predict all the different ways in which users will interact with the dialog agent once it is deployed, therefore designing one is often a long process in which different versions need to be tested and improved. This *learning* process can be long and costly and is, therefore, an important aspect of the DM approach. Rule-based systems have handwritten rules that the program cannot update itself, therefore the learning needs to happen iteratively with manual updates of the interaction rules that can be a hard process and limit the scalability and dialog complexity of such approaches. Rasa Core learns through interactions in which the developer confirms the correctness of the dialog agents' actions, as such it can be seen as a manual process but on an online system. OpenDial has two learning processes, first the probabilistic rules need to be updated which can be done iteratively with supervised learning when new data becomes available. Second, the underlying policy is trained like the POMDP approach with reinforcement learning. This can happen online as well. Reinforcement learning further allows for planning the conversation ahead as future rewards are considered which makes this approach better for complex tasks. Finally, memory networks can be trained end-to-end, which can allow for a simplified learning process without any preprocessing and data labeling. However, online learning has not yet been shown for memory networks.

Error handling helps the dialog agent to handle unexpected inputs, making the conversations robust and natural. In most cases the finite-state and frame-based approach use a simple escalation message such as "I didn't understand" to give feedback to the user, however, users either have to rephrase or start a new interaction. In TrindiKit the developer can define rules for recovering from bad input, while the data-driven and hybrid approaches keep track of the state probability and lower that on unexpected input, returning to previous states in order to build up new evidence for the current belief state. This makes them quite robust as they usually reach their goal. However, unexpected input increases the length of the conversation because turns will be repeated. Handcrafted systems are less elegant in handling such situations. In example-based tools, errors are usually not detected or handled at all, the system just selects the next message to send, which can be hit-or-miss. Finally, error handling in memory networks have not yet been studied in depth.

Regarding *dependencies*, finite-state and frame-based systems are ready to use without any prior data or model, also the underlying state and

policy are easy to understand, thus making these tools very approachable. The research work focused TrindiKit requires the implementation of different rules which is not trivial and will require an underlying model in order to create a working system. For the other data-driven approaches it is obvious that some training data (and meta-data) is required. Example-based approaches require a lot of examples in order to perform human-like chit-chat conversations. Also, PyDial requires a lot of dialogs, in the range of 1000 for a simple task, which can be lowered with the OpenDial architecture, and Rasa Core lowers this figure to about ten for basic interactions. Memory Networks have been shown to work acceptably well with around 500 conversations. The large number of dialogs for the purely data-driven tools means some way of data-collection is required to create a workable solution making them less approachable. The hybrid approaches lower the data requirement far enough to make manual data creation a viable option.

Domain independence is one of the influencing factors for scalability of the overall dialog system. Finite-state approaches have fixed sequences of messages which make them mostly not reusable, whereas frame-based approaches can have sub-modules (e.g., a payment information module that can be shared between multiple applications), although issues of overfitting to training data could limit their applicability. DialogFlow, just like other similar software, have some available predefined modules that can be implemented into a new dialog agent. Rasa Core does not provide this, but a similar thing would theoretically be possible. TrindiKit can offer domain independence by reusing part of a model. PyDial provides domain-independent policy and state-tracker and requires domain dependent data only for NLU and NLG. Again, memory network research work is not mature enough yet to explore domain independence.

The last dimension is *control* so how much influence the developer has on the flow of the conversation at design time. Handcrafted systems offer the most control, in finite-state flow the whole conversation is fully designed, so complete control of the flow is possible, while the other tools offer conceptual control on a slightly more abstract level. Example-based and neural networks, on the other hand, offer no control at all other than selecting the dialogs of the training database. The other tools offer partial control through defining rewards during reinforcement training or manual interactive training.

DISCUSSION AND OUTLOOK

The comparison of DM approaches has shown that many convenient tools are available to create simple but useful dialog agents and a few research work focused solutions that help exploring new data-driven ways of handling dialogs. Below we summarize the evaluation of the current approaches and solutions against the original principles mentioned in *Dimensions and Analysis* section.

Capability of creating natural, robust and complex dialogs. Nowadays creators of dialog systems are balancing between robustness, naturalness, and complexity, seeking for suitable tradeoffs. Systems based on handcrafted rules are very robust but lack naturalness and complexity. In contrast, probabilistic models give a more natural fee, at a cost of robustness. So far, the ambition to create a system supporting complex dialogs performing well across multiple domains remains to be hardly reachable.

Convenience for developers. Tools for creating dialog systems based on handcrafted rules are not new, and ample support exists. Recently developers got in their toolkit tools allowing to build probabilistic and hybrid solutions. Some of these tools do not require a deep understanding of underlying principles, while some do require giving developers fairly wide freedom for experimentation. Developers can build robust dialog systems in a relatively easy fashion; however, such systems are working in a limited domain, where the naturalness is primarily defined by the amount and diversity of training data the developers can afford.

Applicability in commercial environment. The "robustness" is the first commercial priority (beyond entertainment domain), and current approaches and tools allow to build robust solutions working well in limited domains. It suggests that the current state of the technology is mature enough to be commercially viable, primarily providing another medium for customers

to get a service they god before via websites, mobile applications, or emails.

Scalability/reusability in multiple applications. Solutions based on hand-crafted rules lack scalability to wider and bigger domains. Some tools provide dialog policies for some domains (e.g., weather, chit chat) that could be reused by other developers to shorten time to deployment. However, transfer learning techniques for dialog policies are not currently common, and practices similar to the ones used in computer vision (e.g., reuse of neural networks without the last layer) are indeed in demand.

Human-like highly intelligent computer assistant is still not available, but several concrete work directions should be pursued for next-generation dialog managers to increase their impact on digital assistants and dialog agents in general. We identify four areas of interest.

Training Data Availability. Current efforts are severely limited by the lack of large amount, high-quality training data across different domains. While this might not be an issue for big commercial players, there is a clear challenge for academic research: training data generation is a tedious and expensive process. At the same time, issues of diversity (also in terms of the socio-demographic characteristics of the people involved in the creation of training samples) should be considered, to seek fair and inclusive conversations.

Integration. In current DM systems external data sources are accessed through handwritten external actions. However, future digital assistants shall be able discover and select such actions in a dynamic and adaptive fashion, while influencing at the same time their surrounding devices. Understanding how to design dialog managers able to learn and organize new external sources at run-time could dramatically improve their utility. The learning process could be carried through conversations (with chatbot users or domain experts),[8] or even automatically.

Context awareness. Dialog systems are still only able to perform quite simple tasks which often can be performed almost as quickly through using a mobile or web interface. One of the aspects which might make the future assistants really useful is understanding more of the context in which they exist (beyond location information). Using sensors in mobile devices, digital assistants could make suggestions for a restaurant around the user or suggest a movie on TV. Sensing devices might then act as another input for the dialog manager.

Policy generation. In the movies, characters can talk to the assistant about any topic and even learn from them. Creating or even learning dialog policies for any type of conversation or domain will not be a viable option. In order to create a dialog agent that can answer naturally to questions on any topic, there needs to be some automatic policy generation for unknown domains. Transfer learning approaches[16] provide a way to bootstrap dialog policies across domains, but their applicability and performance are still limited.

ACKNOWLEDGMENTS

This work was supported by the Amsterdam Institute for Advanced Metropolitan Solutions with the *AMS Social Bot* grant.

REFERENCES

1. M. McTear, Z. Callejas, and D. Griol, "Introducing the conversational interface," in *The Conversational Interface*. New York, NY, USA: Springer, 2016. Available at: https://doi.org/10.1007/978-3-319-32967-3_1
2. J. Weizenbaum, "ELIZA – A computer program for the study of natural language communication between man and machine," *Commun. ACM*, vol. 23, no.20, pp. 36–45, 1966. Available at: http://dx.doi.org/10.1145/365153.365168
3. K. M. Colby, F. D. Hilf, S. Weber, and H. C. Kraemer, "Turing-like Indistinguishability tests for the validation of a computer simulation of paranoid processes," *Artif. Intell.*, vol. 3, pp. 199–221. Available at: https://doi.org/10.1016/0004-3702(72)90049-5
4. D. Jurafsky, and J. H. Martin, "Speech and language processing - An introduction to natural language processing, computational linguistics, and speech recognition," 2017 Retrieved from https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf
5. S. Young, M. Gašić, B. Thomson, and J. D. Williams, "POMDP-based statistical spoken dialogue systems: A review," *Proc. IEEE*, vol. 101, no. 5, pp. 1160–1179, 2013. Available at: https://ieeexplore.ieee.org/abstract/document/6407655/
6. A. Bordes, Y.-L. Boureau, and J. Weston, "Learning end-to-end goal-oriented dialog," in *Proc. ICLR*, 2016,

pp. 1–15. Available at: https://openreview.net/pdf?id=S1Bb3D5gg
7. P. Lison, "A hybrid approach to dialogue management based on probabilistic rules," *Comput. Speech Lang.*, vol. 34, pp. 232–255, 2015. Available at: https://doi.org/10.1016/j.csl.2015.01.001
8. P. Kucherbaev, A. Bozzon, and G. Houben, "Human aided bots," *IEEE Internet Comput.*, to be published. Available at: https://doi.org/10.1109/MIC.2018.252095348.
9. Introduction to Flow XO, URL: Available at: https://support.flowxo.com/article/132-introduction. Retrieved: Dec. 9, 2017.
10. Basics of DialogFlow, URL: Available at: https://dialogflow.com/docs/getting-started/basics. Retrieved: Dec. 9, 2017.
11. D. Traum, and S. Larsson, "The information state approach to dialogue management," in *Current and New Directions in Discourse and Dialogue*. New York, NY, USA: Springer, pp. 325–353, 2003. Available at: https://doi.org/10.1007/978-94-010-0019-2_15
12. About Chatterbot, URL: http://chatterbot.readthedocs.io/en/stable/, Retrieved: Dec. 9, 2017.
13. S. Ultes *et al.*, "PyDial a multi-domain statistical dialogue system toolkit," in *Proc. ACL Demo*, 2017, pp. 73–78. Available at: https://doi.org/10.18653/v1/P17-4013
14. T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa: Open source language understanding and dialogue management," in *Proc. NIPS Workshop Conversational AI*, 2017, pp. 1–9. Available at: https://arxiv.org/pdf/1712.05181.pdf
15. J. D. Williams, K. Asadi, and G. Zweig, "Hybrid code networks: Practical and efficient end-to-end dialog control with supervised and reinforcement learning," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, 2017, pp. 665–677. Available at: https://doi.org/10.18653/v1/P17-1062
16. M. Gašić *et al.*, "Dialogue manager domain adaptation using Gaussian process reinforcement learning," *Comput. Speech Lang.*, vol. 45, pp. 552–569, 2017. Available at: https://doi.org/10.1016/j.csl.2016.09.003

Jan-Gerrit Harms is an Ms.C. student with the Web Information Systems Group, Delft University of Technology, Delft, The Netherlands. His research interests include ontology learning, conversational agents, and blockchain technology. Contact him at jan.gerrit.harms@gmail.com.

Pavel Kucherbaev is a Postdoctoral Researcher with the Web Information Systems Group, Delft University of Technology, Delft, The Netherlands. His research focuses on human computation and conversational agents. He received the Ph.D. degree from University of Trento, Trento, Italy. Contact him at pavel.kucherbaev@gmail.com.

Alessandro Bozzon is an Associate Professor with the Web Information Systems group, Delft University of Technology, Research Fellow with the AMS Amsterdam Institute for Advanced Metropolitan Solutions, and a Faculty Fellow with the IBM Benelux Center of Advanced Studies. His research interests include the intersection of crowdsourcing, user modeling, and web information retrieval. Contact him at a.bozzon@tudelft.nl.

Geert-Jan Houben is a Full Professor and the Leader with the Web Information Systems research group of TU Delft, a Scientific Director of Delft Data Science, a Research Program Leader on Open & Online Education in TU Delft Extension School, and a Principal Investigator in AMS, Amsterdam Institute for Advanced Metropolitan Solutions. His research group covers subjects in the wider field of web engineering and web science, and his research focuses on user modeling for web-based systems. Contact him at g.j.p.m.houben@tudelft.nl.