OpenDial: A Toolkit for Developing Spoken Dialogue Systems with Probabilistic Rules

Pierre Lison
Language Technology Group
University of Oslo (Norway)
plison@ifi.uio.no

Casey Kennington*
Department of Computer Science
Boise State University (USA)
casey.kennington@cs.boisestate.edu

Abstract

We present a new release of OpenDial, an open-source toolkit for building and evaluating spoken dialogue systems. The toolkit relies on an information-state architecture where the dialogue state is represented as a Bayesian network and acts as a shared memory for all system modules. The domain models are specified via probabilistic rules encoded in XML. OpenDial has been deployed in several application domains such as human–robot interaction, intelligent tutoring systems and multi-modal in-car driver assistants.

1 Introduction

The recent advent of voice-controlled personal assistants (such as Siri, Cortana or Alexa) has popularised the use of speech as a means for interfacing with everyday devices. These dialogue systems are built on complex architectures that include components such as speech recognition, language understanding, dialogue management, generation, speech synthesis, situation awareness and multi-modal inputs/outputs. To allow developers to abstract from implementation details and focus on high-level domain modelling, a number of software frameworks have been developed to glue together these components, such as Olympus/Ravenclaw (Bohus and Rudnicky, 2009), the AT&T Statistical Dialog toolkit (Williams et al., 2010), InproTK (Baumann and Schlangen, 2012) or IrisTK (Skantze and Al Moubayed, 2012).

Existing frameworks can be grouped in two categories. On the one hand, symbolic frameworks rely on finite-state automata or logical methods to represent and reason over the current dialogue state. While they provide fine-grained control over the dialogue, these approaches are often poor at handling errors and uncertainty. On the other hand, statistical frameworks capture the interaction dynamics in a probabilistic manner and seek to optimise the dialogue behaviour from data. However, these methods typically require large amounts of data, making them difficult to apply in domains for which dialogue data is scarce.

In this paper, we present a new release of OpenDial[1], a Java-based, open-source software toolkit designed to facilitate the development of spoken dialogue systems in domains such as personal assistants, in-car driving interfaces, intelligent tutoring systems, or even autonomous robots. OpenDial adopts a hybrid approach that combines the benefits of logical and statistical methods to dialogue modelling into a single framework. The toolkit relies on *probabilistic rules* to represent the internal models of the domain (i.e. the probability and utility models employed to update the dialogue state and make decisions) in a compact and human-readable format. Crucially, the probabilistic rules can contain unknown parameters that can be efficiently estimated from dialogue data using supervised or reinforcement learning.

This paper is structured as follows. Section 2 presents the toolkit architecture and Section 3 explains how to specify dialogue domains with probabilistic rules. Section 4 reviews the toolkit's implementation and Section 5 its deployment in several application domains. Finally, Sections 6 and 7 relate OpenDial with other frameworks and summarise the key contributions of the toolkit.

2 Architecture

OpenDial relies on a information-state architecture (Larsson and Traum, 2000) in which all com-

*The present work was conducted while the author was affiliated to CITEC, Bielefeld University (Germany).

[1] http://www.opendial-toolkit.net

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics—System Demonstrations, pages 67–72,
Berlin, Germany, August 7-12, 2016.

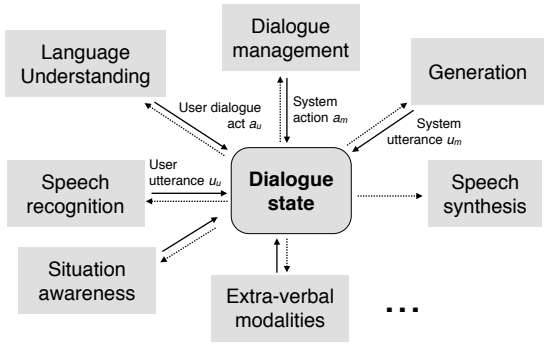


Figure 1: Information-state architecture for the toolkit, with the dialogue state acting as a central shared memory for all system components.

ponents work together on a shared memory that represents the current dialogue state. This dialogue state is factored into distinct variables, each representing a particular aspect of the interaction (e.g. the user intention, the dialogue history or the external context). The dialogue state is encoded as a Bayesian network, which is a directed graphical model where the nodes represent the state variables and the edges are conditional dependencies. The key benefit of this probabilistic representation of the dialogue state is the ability to capture uncertainties and partially observable variables, which are commonplace in most dialogue domains.

Figure 1 illustrates the general architecture. The dialogue system is composed of a set of components which are continuously monitoring the dialogue state for relevant changes. When such a change occurs, the corresponding modules can react to such events and further modify the dialogue state, thereby generating new updates. A typical information flow starts with the speech recogniser, which periodically outputs recognition hypotheses u_u.[2] Language understanding maps these hypotheses into representations of the dialogue act a_u expressed by the user. Dialogue management then selects the system action a_m to perform. If the selected action is a communicative act, language generation is triggered to find its linguistic realisation, denoted u_m. Finally, the constructed utterance is sent to the speech synthesiser.

OpenDial provides two ways to integrate new components into a dialogue system. The first is to specify a *model*, which is a collection of probabilistic rules (see next section). Each model is associated with one or more *trigger variables*, i.e. variables that trigger the instantiation of the rules upon their update. Alternatively, one can also implement an external module from scratch and connect it to the dialogue state. OpenDial provides a Java API to easily integrate such external modules, which may either wait for update events from the dialogue state or run asynchronously.

[2] We denote user-specific variables with the subscript u and machine-specific variables with the subscript m.

3 Dialogue domains

OpenDial is fully domain-independent. To apply it to a particular domain, the system developer provides a specification of the dialogue domain encoded in XML. This XML file contains the following information:

1. The initial dialogue state;
2. A collection of domain models, which are themselves composed of probabilistic rules;
3. Prior distributions for unknown parameters in the probabilistic rules (if any);
4. Optional configuration settings.

3.1 Probabilistic rules

The probabilistic rules in the domain models are expressed as *if...then...else* constructions that map logical conditions on some state variables to probabilistic effects on some other state variables. Due to space constraints, we only provide here a brief overview of the formalism, the reader is invited to consult Lison (2015) for more details.

The rule conditions are expressed as logical formulae, using the usual logic operators (conjunctions, disjunctions, and negations) and various binary relations (equality, inequalities, string matching, etc.). The conditions may also include underspecified (i.e. free) variables which are universally quantified on the top of the rule. Each condition is associated with a distribution over mutually exclusive effects, where each effect is an assignment of values to some state variable(s). Here is a simple example of probabilistic rule:

$$\forall x, \textbf{if } (a_u = \mathit{Request}(x) \wedge a_m = \mathit{Verify}(x)) \textbf{ then } \Big\{ P(a_u' = \mathit{Confirm}(x)) = 0.9$$

The rule expresses a prediction on the future user dialogue act a_u' based on the last user dialogue act a_u and system's action a_m. The rule stipulates that if the user requested some x and the system replied

by asking whether the request is indeed x, the user is expected to comply and confirm their request with probability 0.9. A void effect with no prediction is implicitly associated with the remaining probability mass (0.1 here). The universal quantification on x indicates that this probability is independent of the type of user request.

The *if...then...else* structure of the probabilistic rules partitions the state space into groups of similar states (corresponding to the logical conditions). In particular, the sequential ordering of the conditions enable dialogue developers to write rules with "backoff strategies", starting from the most specific conditions and then gradually moving to more generic cases if the top conditions do not apply. Such partitioning of the state space is important to ensure the probabilistic rules are able to generalise to new, unseen situations.

Probabilistic rules can express both conditional probability distributions and utility functions. At runtime, the rules are instantiated as latent nodes in the Bayesian network representing the dialogue state. The rules can therefore be seen as high-level templates for the construction of directed graphical models (Lison, 2015). The latest release of OpenDial offers several new functionalities such as the support for custom functions and the ability to directly manipulate relational structures – such as dependency trees, semantic graphs or hierarchical task networks – in the probabilistic rules.

3.2 Example

Listing 1 provides a simple example of dialogue domain in XML. The domain specifies the behaviour of a elevator that can move between three floors through a voice-controlled interface. The interaction starts with a system prompt ("*Which floor do you want?*") followed by the user request (e.g. "*third floor, please*"). If the request is uncertain, the elevator should ask the user to confirm (e.g. "*Do you want the third floor?*").

The domain contains an initial state with one variable (the initial prompt) and three models: an intent recognition model mapping the user utterance u_u to the corresponding dialogue act a_u, an action selection model encoding the utility of the system actions a_m, and a third model responsible for generating the system responses u_m and predicting the next user act a'_u. Each model is associated with a trigger variable and is composed of a set of probabilistic rules. The rules are written

```
<domain>
  <initialstate>
    <variable id="u_m">
      <value prob="1">Which floor do you want?</value>
    </variable>
  </initialstate>

  <!-- Intent recognition -->
  <model trigger="u_u">
    <rule>
      <case>
        <condition operator="and">
          <if var="X" relation="in" value="[first,second,third]"/>
          <if var="u_u" relation="contains" value="{X} (floor)?"/>
        </condition>
        <effect prob="1">
          <set var="a_u" value="Request({X})"/>
        </effect>
      </case>
      <case>
        <condition operator="and">
          <if var="u_u" relation="contains" value="(yes|exactly)"/>
          <if var="a_m" relation="=" value="Verify({X})"/>
        </condition>
        <effect prob="1">
          <set var="a_u" value="Confirm({X})"/>
        </effect>
      </case>
    </rule>
  </model>

  <!-- Action selection model -->
  <model trigger="a_u">
    <rule>
      <case>
        <condition operator="or">
          <if var="a_u" relation="=" value="Request({X})"/>
          <if var="a_u" relation="=" value="Confirm({X})"/>
        </condition>
        <effect util="1">
          <set var="a_m" value="GoTo({X})"/>
        </effect>
        <effect util="0.5">
          <set var="a_m" value="Verify({X})"/>
        </effect>
      </case>
      <case>
        <effect util="-2">
          <set var="a_m" value="GoTo({X})"/>
        </effect>
      </case>
    </rule>
  </model>

  <!-- Generation and user action models -->
  <model trigger="a_m">
    <rule>
      <case>
        <condition>
          <if var="a_m" relation="=" value="GoTo({X})"/>
        </condition>
        <effect util="1">
          <set var="u_m" value="Ok, going to the {X} floor"/>
        </effect>
      </case>
      <case>
        <condition>
          <if var="a_m" relation="=" value="Verify({X})"/>
        </condition>
        <effect util="1">
          <set var="u_m" value="Do you want the {X} floor?"/>
        </effect>
      </case>
    </rule>

    <rule>
      <case>
        <condition operator="and">
          <if var="a_m" relation="=" value="Verify({X})"/>
          <if var="a_u" relation="=" value="Request({X})"/>
        </condition>
        <effect prob="0.9">
          <set var="a_u^p" value="Confirm({X})"/>
        </effect>
      </case>
    </rule>
  </model>

</domain>
```

Listing 1: Dialogue domain example in XML.

ten as sequences of *if-then-else* cases, where each case has a (possibly void) condition and a set of corresponding effects. Curly brackets such as {X} denote underspecified variables.

Intent recognition contains one single rule which maps utterances matching the pattern "x *(floor)?*" where $x \in$ ["first", "second","third"] to the dialogue act Request(x), and maps the responses "*yes*" or "*exactly*" following the system action Verify(x) to the dialogue act Confirm(x). This rule is deterministic, since all its effects have a probability 1 if their condition is met. A default value is assigned to a_u if no condition applies.

The action selection model expresses the utility of two system actions: GoTo(x), representing the action of moving to the floor x, and the clarification Verify(x). The two actions respectively have a utility of 1 and 0.5 if the last user act is Request(x) or Confirm(x). Otherwise, the action GoTo(x) has a negative utility of -2. The action GoTo(x) will therefore be selected if the probability of the user act Request(x) is higher than 0.8, while Verify(x) will be chosen if this probability is lower.

The generation model simply maps the system actions to their corresponding surface realisations.[3] Finally, the prediction model (corresponding to the example in the previous section) states that the probability of the user confirming their request when asked to do so is set to 0.9.

The example could of course be extended in many ways – for instance by explicitly specifying the current floor as state variable, and providing prior distributions on the floor requested by the user, conditioned on the current one. The user guide on the OpenDial website provides several additional examples of dialogue domains.

3.3 Parameter Estimation

The probabilistic rules in the example were associated with fixed probabilities or utilities. However, in most domains, these values are difficult to determine in advance and are best learned from empirical data. The toolkit allows dialogue developers to estimate unknown parameters via Bayesian learning. In practice, this is done by replacing the probability or utility values in the rules by parameters associated with prior distributions. For instance, the utilities 1, 0.5 and -2 in the action selection model can be replaced by three Gaussians representing the spread of possible utility values, and the probability 0.9 in the prediction rule can be replaced by a Dirichlet expressing the prior belief about the user response. System developers can then exploit dialogue data to automatically learn the posterior distributions for these parameters. Two methods have been developed: supervised learning from Wizard-of-Oz data (Lison, 2015) and reinforcement learning from real or simulated experience (Lison, 2013). Both learning methods assume that the rule structure – the mapping between conditions and effects – is provided by the developer, while the numerical parameters are determined via statistical estimation. Indeed, system developers often have a good grasp of the general structure of the dialogue domain but are typically unable to quantify the precise probability of a prediction or utility of an action.

4 Implementation

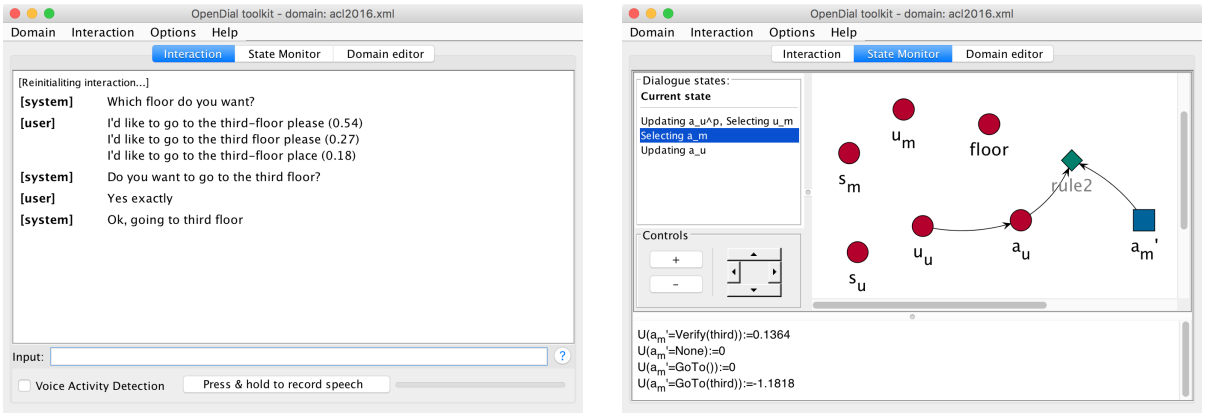
OpenDial is implemented in Java and is released under an open-source MIT license. The software comes along with a graphical user interface which allows developers to run a given dialogue domain and test its behaviour in an interactive manner. Three views are available in the interface. The "chat window" view, shown in Figure 2(a), displays the dialogue history and let the user enter new (text or speech) inputs. In the "dialogue monitor" view, shown in Figure 2(b), the user can inspect the Bayesian network representing the current dialogue state, perform various inference queries (e.g. calculating marginal distributions), and look at previous state updates. This last feature is particularly useful for debugging. Finally, the "domain editor" view provides an interactive editor for the XML domain file.

To ensure the system can quickly react to new events, most processes operate in anytime mode, which implies they can be gracefully interrupted and deliver their outputs at any time. Both exact and approximate inference are employed to update the dialogue state and plan system actions.

A collection of plugins extends the toolkit with additional modules. Plugins are notably available to connect OpenDial to Nuance and AT&T cloud-based speech APIs, to the MaltParser for data-driven dependency parsing, the Sphinx speech recogniser and the MARY speech synthesiser.[4]

[3] In a real elevator, an external module will of course need to convert the actions GoTo(x) into a physical motion.

[4] See http://developer.nuance.com, http://developer.att.com/apis/speech (to be closed), http://cmusphinx.sourceforge.net,



(a) Chat window

(b) Dialogue state monitor

Figure 2: Graphical user interface for OpenDial.

5 Application Domains

OpenDial has been deployed in several application domains, either directly as an end-to-end dialogue system, or as a specific component in a larger software architecture, typically to handle dialogue state tracking and management tasks.

An important application domain is human–robot interaction. Lison (2015) illustrates how OpenDial is used in a human–robot interaction domain where a humanoid robot is instructed to navigate through a simple environment, fetch an object and bring it to a particular landmark. The experiment shows in particular how the parameters of probabilistic rules can be efficiently learned from limited amounts of Wizard-of-Oz data.

OpenDial was used in another human–robot interaction domain with multiple human participants. Kennington et al. (2014) describe how OpenDial was employed as the primary dialogue manager in a twenty-questions game scenario between a robot and up to three human participants. Using Wizard-of-Oz data, the parameters were estimated with the toolkit's training regime. During evaluation, multiple instantiations of OpenDial were used to model the interaction with each participant. Even though the instantiations were mutually independent, all shared the same modules, allowing communication between them when turn-taking decisions needed to be made.

OpenDial was also deployed as a dialogue manager in an in-car dialogue scenario (Kousidis et al., 2014). The objective was to deliver upcoming calendar entries to the driver (e.g. "*on Tuesday you have lunch with John at the cafeteria*")

Figure 3: Driver's view from the OpenDS driving simulator [http://www.opends.eu].

via speech and the toolkit was employed to determine when the speech should be interrupted. The driver could also indicate to the system via speech or a head nod that the interrupted speech should continue. Information from the simulated driving environment (see Figure 3) was used to make the system "situationally aware" and able to react to dangerous events by interrupting speech, allowing the driver to focus on the primary task of driving. OpenDial was employed to dynamically track the state of the dialogue system over time.

6 Discussion

The purpose of OpenDial is to combine the expressivity of logic-based frameworks with the robustness and adaptivity of statistical systems. In line with logic-based frameworks (Larsson and Traum, 2000; Bohus and Rudnicky, 2009), the toolkit provides system developers with powerful abstractions to structure the domain models, since probabilistic rules can make use of complex logical formulae and universal quantification. And in line with statistical approaches (Young et al., 2013),

http://www.maltparser.org and http://mary.dfki.de.

OpenDial is also able to explicitly handle uncertain knowledge and stochastic relations between variables thanks to its probabilistic representation of the dialogue state and its ability to estimate unknown parameters from data.

The presented framework is very general and can be employed to design a wide spectrum of models, from traditional handcrafted models (such as finite-state automata) on one extreme to probabilistic models fully estimated from data on the other extreme. The choice of model within this spectrum boils down to a design decision based on the relative availabilities of training data and domain knowledge. Furthermore, OpenDial enables users to quickly develop a working system with little or no data, then gradually extend and refine their models as more data becomes available.

The primary focus of OpenDial is on high-level processes such as language understanding, dialogue management and generation. Compared to frameworks such as IrisTK (Skantze and Al Moubayed, 2012) or InproTK (Baumann and Schlangen, 2012), OpenDial offers more limited support for lower-level interaction control such as attentional behaviours or turn-taking strategies. How to reconcile the "low-level" and "high-level" aspects of dialogue modelling in a principled manner is an important question for future work.

7 Conclusion

We presented a new release of OpenDial, a Java-based toolkit for developing and evaluating spoken dialogue systems. The toolkit rests on a hybrid modelling framework that seeks to combine the benefits of logical and probabilistic approaches to dialogue. The dialogue state is represented as a Bayesian network, and the domain models are specified using probabilistic rules. Unknown rule parameters can be automatically estimated from dialogue data using Bayesian learning.

OpenDial is in our view particularly well-suited to handle dialogue domains that exhibits both a complex state-action space and high levels of noise and uncertainty. Typical examples of such dialogue domains are human-robot interaction, virtual assistants and tutoring systems. These domains must often deal with state-action spaces that include multiple tasks to perform in rich interaction contexts. They are also confronted with substantial levels of uncertainty, arising from speech recognition errors and partially observable environments. Due its hybrid modelling approach, the toolkit is able to capture such dialogue domains in a relatively small set of probabilistic rules and associated parameters, allowing them to be tuned from modest amounts of training data, which is a critical requirement in many applications.

Source code and documentation

The www.opendial-toolkit.net website presents the release along with the source code and a step-by-step user guide. A screencast is also available at https://www.youtube.com/watch?v=X8x8Qj5Z7Ag.

References

T. Baumann and D. Schlangen. 2012. The InproTK 2012 release. In *NAACL-HLT Workshop on Future Directions and Needs in the Spoken Dialog Community*, pages 29–32, Montréal, Canada.

D. Bohus and A. Rudnicky. 2009. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3):332–361.

C. Kennington, K. Funakoshi, Y. Takahashi, and M. Nakano. 2014. Probabilistic multiparty dialogue management for a game master robot. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 200–201.

S. Kousidis, C. Kennington, T. Baumann, H. Buschmeier, S. Kopp, and D. Schlangen. 2014. A Multimodal In-Car Dialogue System That Tracks The Driver's Attention. In *Proceedings of ICMI*, Istanbul, Turkey.

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. 2013. Model-based Bayesian reinforcement learning for dialogue management. In *Proceedings of the 14th Annual Conference of the International Speech Communication Association*, Lyon, France.

P. Lison. 2015. A hybrid approach to dialogue management based on probabilistic rules. *Computer Speech & Language*, 34(1):232 – 255.

G. Skantze and S. Al Moubayed. 2012. IrisTK: A statechart-based toolkit for multi-party face-to-face interaction. In *Proceedings of the 14th International Conference on Multimodal Interaction (ICMI 2012)*, pages 69–76, New York, USA.

J. Williams, I. Arizmendi, and A. Conkie. 2010. Demonstration of AT&T Let's Go: A production-grade statistical spoken dialog system. In *Proceedings of the the IEEE Spoken Language Technology Workshop*, pages 157–158.

S. Young, M. Gasic, B. Thomson, and J. Williams. 2013. POMDP-based statistical spoken dialogue systems: A review. *Proceedings of the IEEE*, PP(99):1–20.