

A methodology for turn-taking capabilities enhancement in Spoken Dialogue Systems using Reinforcement Learning[☆]

Hatim Khouzaimi^{a,b,1,*}, Romain Laroche^{a,2}, Fabrice Lefèvre^b

^a Orange Labs, Châtillon, France

^b CERI-LIA, Avignon, France

Received 23 August 2016; received in revised form 30 June 2017; accepted 7 July 2017

Available online 20 July 2017

Abstract

This article introduces a new methodology to enhance an existing traditional Spoken Dialogue System (SDS) with optimal turn-taking capabilities in order to increase dialogue efficiency. A new approach for transforming the traditional dialogue architecture into an incremental one at a low cost is presented: a new turn-taking decision module called the *Scheduler* is inserted between the Client and the Service. It is responsible for handling turn-taking decisions. Then, a User Simulator which is able to interact with the system using this new architecture has been implemented and used to train a new Reinforcement Learning turn-taking strategy. Compared to a non-incremental and a handcrafted incremental baselines, it is shown to perform better in simulation and in a real live experiment.

© 2017 Elsevier Ltd. All rights reserved.

Keywords: Spoken Dialogue Systems; Turn-taking; Incremental dialogue; Reinforcement Learning

1. Introduction

Floor management in currently deployed dialogue systems is generally pretty basic: the user and the system must wait for each other to finish their respective utterances before taking the floor at each turn. This way of handling turns has the advantage of simplicity and maintainability. However, during the last 15 years, an important research thread has shown that this is not an optimal manner of managing turn-taking in human/computer dialogue (Aist et al., 2007; Skantze and Schlangen, 2009; El Asri et al., 2014) and that incremental³ dialogue systems (Schlangen and Skantze, 2011; El Asri et al., 2014) offer a better user experience. A dialogue system is incremental when it is able to process the user's utterance as it is spoken which makes it able to take the floor anytime during the dialogue. Also, the user

[☆] This paper has been recommended for acceptance by Prof. R. K. Moore.

* Corresponding author:

E-mail address: hatim.khouzaimi@askhub.io (H. Khouzaimi), romain.laroche@maluuba.com (R. Laroche).

¹ AskHub, Paris, France.

² Maluuba, Montreal, Canada.

³ From an architectural point of view, incrementality refers to the ability to process increments separately. On the other hand, from the functional point of view, it designates the ability to process and act on speech as it is spoken. In this paper, we use the second frame since it is a feature that systems must have in order to improve their turn-taking capabilities.

is allowed to interrupt the system. The concept of incremental processing has been proposed for the first time to build incremental compilers (Lock, 1965). According to Kilger and Finkler (1995), it has been introduced in the field of natural language processing in Wirén (1992). The idea of building incremental dialogue systems is directly inspired by human conversation since when people talk to each other, the listener most often tends to understand the speaker as she speaks even guessing the rest of her sentence before its end (Levelt, 1989; Tanenhaus et al., 1995).

In order to meet industrial constraints (Pieraccini and Huerta, 2005; Paek and Pieraccini, 2008; Laroche, 2010; Asri, 2016), in this paper, a new methodology for turn-taking capabilities enhancement at a low cost is proposed.

During the last decade, dialogue systems have been used to solve various tasks already, both in research and industry. As far as incremental dialogue systems are concerned, most of them are built from scratch with an initial intention of providing them with incremental capacities (Dohsaka and Shimazu, 1997; Allen et al., 2001; Schlangen and Skantze, 2011). In this paper, a new approach for transforming a traditional dialogue system into an incremental one at a low cost is introduced. A new turn-taking decision module is added to the traditional architecture (without modifying the dialogue manager): the *Scheduler*, firstly introduced in Khouzaimi et al. (2014a; 2014b).

In the field of dialogue systems, collecting data corpora is very costly. Therefore, user simulation techniques are widely used (Eckert et al., 1997; Schatzmann et al., 2006; Pietquin and Hastie, 2013; Laroche and Genevay, 2016). In this work, an incremental User Simulator is described (Khouzaimi et al., 2016). It is based on a new approach that is aimed to generate ASR instability (Selfridge et al., 2011). The implemented task is a slot-filling personal agenda management.

Since its first application to dialogue management (Levin and Pieraccini, 1997; Singh et al., 1999), RL (Sutton and Barto, 1998) has become one of the leading frameworks in the field. Here, it is applied to learn turn-taking decisions by taking dialogue duration and task completion as the only components of the reward function (Khouzaimi et al., 2015a), resulting in significantly more efficient and more robust dialogue systems. Compared to other techniques using supervised learning (Meena et al., 2013), no labelling effort is required here. Also, there is no need to make assumptions like the fact that the system should minimise gaps and overlaps (Sacks et al., 1974); the global dialogue quality is the only function to maximise.

Finally, a live experiment where users interact with a smart home (application called the Majordomo) has been run in order to validate this approach. It is shown that the RL strategy significantly improves the task completion ratio. Also, the subjective evaluation is slightly in favour of this new strategy.

Section 2 describes the related work and the positioning of this paper, then Section 3 presents the new methodology for transforming a traditional dialogue system into an incremental one. Based on that, Section 4 describes the simulated environment and the implementation of the rule-based turn-taking strategy. Finally, Section 5 introduces the RL model, Section 6 describes the live experiment as well as the associated results and Section 7 concludes and sheds light on planned future work.

2. Related work

Improving dialogue systems' turn-taking capacities has been an active research thread during the last two decades. Existing contributions can be classified in three design categories (handcrafted, supervised learning and RL) as well as two evaluation categories (indirect and direct) which results in six system categories. They are depicted in the following, given the turn-taking model they use and the way they evaluate it. This idea is synthesised in Table 1.

Some papers use rule-based models or supervised learning that are not directly evaluated with users through real interactions. Aist et al. (2007) introduce a handcrafted incremental setup that improves user satisfaction while reducing dialogue duration. Nevertheless, pre-recorded utterances that are perfectly understandable by the system were

Table 1
Related work classification according to the model and the evaluation method.

	Handcrafted	Supervised learning	RL
Indirect evaluation	Aist et al. (2007)	Meena et al. (2013); Zhao et al. (2015)	Jonsdottir et al. (2008); Selfridge and Heeman (2010); Lu et al. (2011); Dethlefs et al. (2012); Kim and Banchs (2014)
Direct evaluation	Raux and Eskenazi (2009); Skantze and Schlangen (2009); Ghigi et al. (2014); Zhao et al. (2015)	Paetzel et al. (2015)	This paper

used for evaluation; a few judges are then given videos of the interaction to assess. Meena et al. (2013) explore the capacity of the system to deliver feedback and backchannel. Supervised learning is used to determine the right timing for such behaviours and similarly to the previous paper, dialogues were indirectly evaluated by external judges. Supervised learning is also used in Zhao et al. (2015) in order to determine the right timing for the system to take the floor.

Other papers also use handcrafted and supervised learning based strategies but they are directly evaluated through real interactions. A state machine model is introduced in Raux and Eskenazi (2009) where the objective is to minimise speech gaps and overlaps between the system and the user. A handcrafted cost function is used to implement the turn-taking strategy which is shown to reduce the system's latency compared to a fixed silence threshold strategy, while preventing it from interrupting the user too early. Skantze and Schlangen (2009) introduce an incremental dialogue system which asks the user to dictate a number. Using a handcrafted feedback and backchannel strategy for acknowledgement and grounding, it shows that the incremental version of the system improves user satisfaction (even though dialogue duration remains the same). Also, in Ghigi et al. (2014) and (Zhao et al., 2015), handcrafted strategies are tested with real users in a bus scheduling domain. They are able to actively interrupt the user and they are shown to improve the task completion ratio. Finally, Paetzel et al. (2015) compare several turn-taking strategies with different degrees of incrementality. Users were given several tasks to accomplish during a limited amount of time with an incentive which is proportional to the number of tasks accomplished. Users were more satisfied with the fully incremental strategy since it is the more reactive hence making possible to maximise their incentive, nevertheless, this introduces a certain bias since in most real dialogue situations, users are not subject to this time pressure. The system was triggered to take the floor once the Natural Language Understanding (NLU) score reaches a certain threshold. Another dialogue duration threshold above which it decides to abandon the current task and move to the next is also used and they are both learnt from data (previously collected corpus).

Finally, some papers use RL models and indirect evaluation (simulation and offline technique). Jonsdottir et al. (2008) use prosody features exclusively (semantic content is not taken into account) to learn a smooth turn-taking strategy. The system learns in a self-play fashion and the resulting dialogues are then compared with human conversations. In Selfridge and Heeman (2010), a model where the speaker that has the most interesting information to share is the one to take the floor. RL is used in a simulated environment in order to achieve shorter dialogues with a better task completion ratio. A simple configuration is used in Lu et al. (2011) in order to learn the right moment to take the floor in a sentence using Partially Observable Markov Decision Processes (POMDPs) and a restricted set of 50 utterances is used. Another approach using Hierarchical RL is used in Dethlefs et al. (2012) where the objective is to optimise both traditional dialogue management (what to say) and turn-taking (when to say it). A few events are responsible for generating rewards like task completion, misunderstandings and the notion of *information density* introduced in the paper. Afterwards, judges are provided with a few utterances with three locations for barge-in and they have to assess which option would be more appropriate. One of them is chosen by the learnt strategy and the other two correspond to baseline strategies. It is shown that judges agree more with the new learnt strategy. Finally, Kim and Banchs (2014) use Inverse RL in order to imitate the behaviours from a human dialogue corpus. The new strategy is evaluated both in simulation and offline (test corpus).

In this paper, we introduce a new RL-based strategy where the only reward is a combination of dialogue duration and task completion (delivered at the end of each independent dialogue task). Therefore, no assumption about what should the dialogue look like to be more efficient is made (like minimising gaps and overlaps for example) and unlike supervised learning approaches, no labeling effort is required. Moreover, to our knowledge, this is the first contribution where an RL-based turn-taking strategy is directly evaluated in a real live experiment. It is shown that the task completion ratio is significantly improved while improving subjective metrics. Local metrics are also assessed and it is shown that this new strategy is more reactive without harming the dialogue by often interrupting the user too early.

Finally, it is noteworthy that the features used here are focused on semantics. A more classic approach which has been thoroughly studied (Jonsdottir et al., 2008; Skantze and Schlangen, 2009; Raux and Eskenazi, 2012; Meena et al., 2013) mainly uses acoustic features with the purpose to make the system perform feedbacks and backchannels and detect utterances endpoints in order to be more reactive. Here, in addition to these capabilities, the system is also able to interrupt the user because it incrementally parses the content of the user utterance. This is a complementary approach which we plan to enrich with acoustics (in Raux and Eskenazi (2012); Meena et al. (2013), combining

acoustic features with content is shown to improve the overall performance of turn-taking strategies). Since generating acoustic data artificially is a hard problem in itself, this part is left for future work.

3. The Scheduler: a turn-taking manager

This section introduces a method for adding advanced turn-taking capabilities to a traditional dialogue system at a low cost (Khouzaimi et al., 2014a; 2014b). First an overview of the proposed new architecture is given, then time-sharing between the user and the system is formalised. Based on that, the implementation details regarding this new approach are presented.

3.1. A new architecture

Traditional spoken dialogue systems are designed as a chain of five modules: Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), Dialogue Management (DM), Natural Language Generation (NLG) and Text-To-Speech (TTS, also known as speech synthesis). Most current incremental dialogue systems respect this architecture where some of the modules, if not all of them, are incremental (Schlangen and Skantze, 2011). An incremental module is able to process incomplete input therefore producing incomplete output.

These systems are mostly built from scratch. Here, the introduced method is aimed to transform an existing traditional dialogue system into an incremental one at a low cost. Since making all the modules (or some of them, at least) incremental in order to use the previous architecture is costly, an alternative approach is adopted.

In our view, the modules are split in two groups, those constituting the Client and those constituting the Service. The Client is the application that captures the user's speech and outputs a synthetic voice. As a consequence, it necessarily contains the ASR and the TTS. The Service is in charge of managing the dialogue from the system's side, therefore, it necessarily contains the DM. The NLU and the NLG can be put on both sides, depending on the designer's choice.

Then, an intermediate module called the *Scheduler* is inserted between the Client and the Service as shown in Fig. 1. From the Client's perspective, the set {Scheduler + Service} should act incremental.

The role of the Scheduler is to manage turn-taking decisions. It decides when the system should take the floor and when it should just listen and wait for further information. To do so, not only the signal coming from the user is taken into account but also information coming from the Service.

Before tackling the functioning of the Scheduler and its implementation, some clarifications about the way the user and the system share time during the interactions are made.

3.2. Time sharing formalisation

In order to clearly define how the Scheduler handles inputs, the way time is handled and shared between the user and the system is formalised in the following. To do so, a set of conditions (Boolean variable) will be used. The exact moment when a condition goes from *false* to *true* is called the *activation time* of that condition.

From the incremental dialogue processing standpoint, the system permanently processes a continuous stream of data. The user's dialogue turn ends at the activation time of a certain condition, here denoted as *EndTurnCond*,

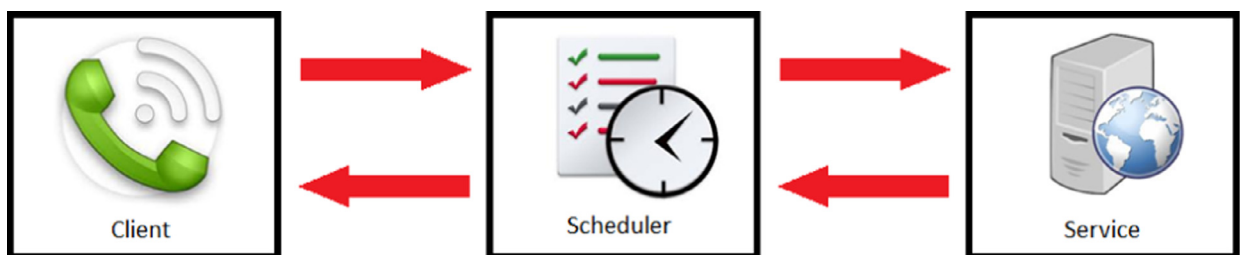


Fig. 1. The Scheduler: an interface between the Client and the Service.

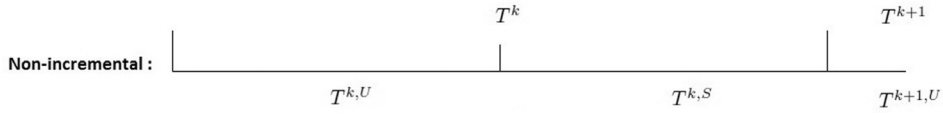


Fig. 2. Traditional time sharing between the user and the system.

which is generally a long enough silence. After that the user gets a response from the system. A *dialogue turn* is the time interval during which the user sends a request and gets a response.⁴ The consecutive dialogue turns will be called T^1, T^2, \dots, T^k , etc. During T^k , the user sends the request Req^k and receives the corresponding response $Resp^k$. The time interval during which Req^k is made is called the *user turn* and noted $T^{k,U}$. Similarly, $Resp^k$ is made during the *system turn* noted $T^{k,S}$. Therefore, $T^k = T^{k,U} \cup T^{k,S}$. This formalisation is illustrated in Fig. 2.

In the case of incremental dialogue, several *micro-turns* compose a single dialogue turn. Just like *EndTurnCond* marks the end of user turns, activation times of *EndMicroTurnCond* delimit *user micro-turns*. For example, let t be the time elapsed from the beginning of the current user turn. A good example of value for *EndMicroTurnCond* would be $\frac{t}{\Delta} \in \mathbb{N}$.

This way, *EndMicroTurnCond* is activated every Δ (e.g., 500 ms). In the case of textual interfaces, it can be activated every time the user hits the space bar for example (a micro-turn corresponding to the user typing a new word). In addition, *EndTurnCond* \Rightarrow *EndMicroTurnCond* but not the other way around and as a consequence, the user turn $T^{k,U}$ is divided into $n^{k,U}$ user micro-turns $\mu T_i^{k,U}$: $T^{k,U} = \bigcup_{i=1}^{n^{k,U}} \mu T_i^{k,U}$. At the end of the i th user micro-turn, only a *partial request* Req_i^k is available which corresponds to the partial utterance that has been uttered so far, during the *partial turn* $T_i^{k,U} = \bigcup_{j=1}^p \mu T_j^{k,U}$. Let us call $ASR(x)$ the ASR output (which is an N -Best) corresponding to request x . It is important to note that if $i_1 < i_2$, then the best hypothesis of $ASR(Req_{i_1}^k)$ is not necessarily a prefix of the best hypothesis of $ASR(Req_{i_2}^k)$. This problem is called *ASR instability* (Selfridge et al., 2011; Khouzaimi et al., 2016) as it is mainly due to ASR and will be tackled later in this paper.

Similarly, a system turn T^k can also be split into $n_{k,S}$ *system micro-turns*: $T^{k,S} = \bigcup_{i=1}^{n_{k,S}} \mu T_i^{k,S}$, a partial response $Resp_p^k$ being what the TTS has output at the end of the *partial system turn* $T_p^{k,S} = \bigcup_{i=1}^p \mu T_i^{k,S}$. The way these micro-turns are delimited is system dependent. For example, when the system enumerates several options that the user should choose from (as in El Asri et al. (2014)), a system micro-turn may correspond to the time interval during which a single option is uttered and thus have a variable duration during a dialogue session.

3.3. The Scheduler implementation

At each new user micro-turn, the current user partial utterance is sent by the Client to the Scheduler. The first decision the latter makes is whether to relay it to the Service or not. The condition controlling this decision is called *ServiceReqCond*. A simple implementation of this condition at user micro-turn k might be: $ServiceReqCond = (Req^k \neq Req^{k-1})$, hence avoiding to send the same request to the Service twice as the expected result is the same. If this condition is true, then the Scheduler sends the whole partial utterance available so far and not only the last portion captured during the last micro-turn. This way of handling incremental processing is called *restart incremental mechanism* in Schlagen and Skantze (2011). The Service's response is then stored by the Scheduler.

Next decision made by the module is whether to relay the Service's response to the Client (activating the TTS) or not. If it decides to do so, it is said to *commit* to the last user's partial request and the associated condition is called *CommitCond* (its activation time is the moment when the Scheduler decides to commit). If not, that means it is still waiting for further information, letting the user complete his request without interrupting him. In fact, the Service is requested most of the time through incomplete requests just to *probe what would be its response*. Suppose the user's request is *Check my account* and the Client sends the partial results *Check*, then *Check my* and finally *Check my account*. The first two requests will be sent to the Service but its response will be *I don't understand*. As a consequence, the Scheduler will most likely decide not to commit to those partial requests and wait for the user to

⁴ This convention is mainly adapted to user initiative interactions (the user asks and the system responds) but it is kept even in the system initiative (the system asks and the user responds) and the mixed initiative (both can ask and respond) cases, even though the denominations *request* and *response* are no longer accurate.

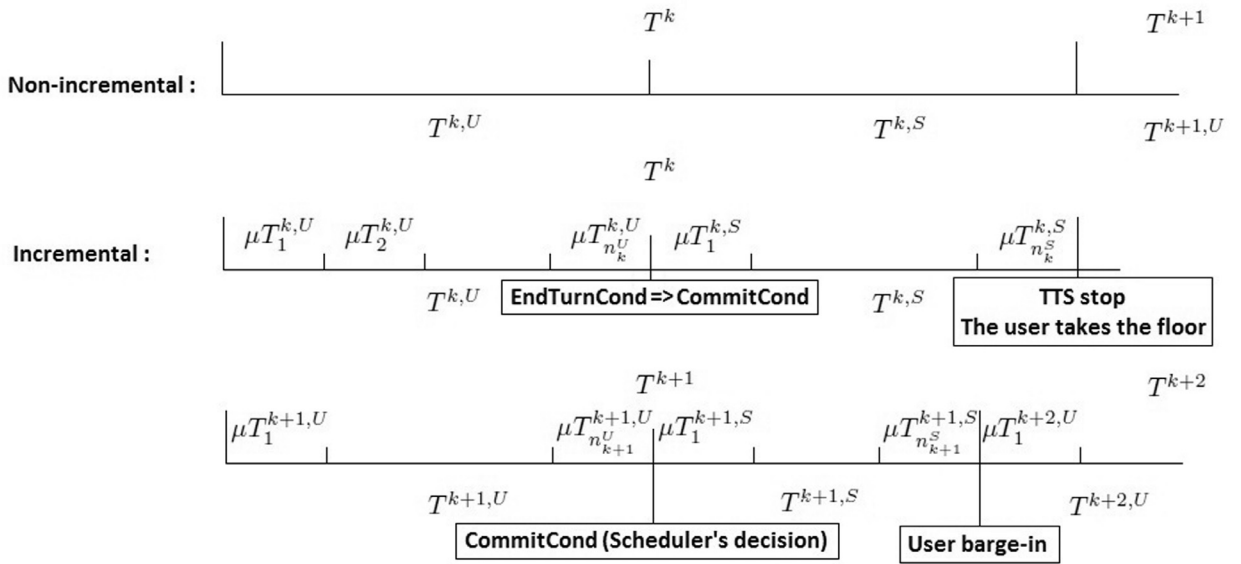


Fig. 3. Time sharing between the user and the system in incremental dialogue systems (the second line is a continuation of the first one).

complete his request. Alternatively, the Client can also force the Scheduler to commit by sending a *signal_ETC* if *EndTurnCond* becomes true. Therefore, $EndTurnCond \rightarrow CommitCond$.

A technical problem remains to be solved: suppose that the Service has a context C just before receiving $ASR(Req_p^k)$ and a new context C' after this request is processed. If the Scheduler does not commit at that point and the user keeps injecting new information then a new request $ASR(Req_{p+1}^k)$ is sent to the Service. Since the restart incremental mode is adopted, Req_{p+1}^k is not a whole new request but an extension of Req_p^k and they are both aimed to be sent to the Service with context C (while the context is C' for the second here).

Thus the Service is modified in order to handle two contexts: the *real context* (C_r) and the *simulation context* (C_s) (or temporary buffer). The commit operation is defined as $Commit : C_r = copy(C_s)$. Processing the request can lead to the modification of C_s only, hence becoming C'_s . However, if the Scheduler does not decide to commit and before sending $ASR(Req_{p+1}^k)$, it performs a *cancel* (or *rollback*) as $Cancel : C_s = copy(C_r)$.

Fig. 3 summarises the time sharing formalisation as well as the different scenario that might lead to turn transition:

- *System to user transition*: The user can wait for TTS to stop before taking the floor or barge-in before that.
- *User to system transition*: The Scheduler can spontaneously decide to take the floor if *CommitCond* or wait for a *signal_ETC* from the Client (since $EndTurnCond \rightarrow CommitCond$)

In this new architecture, the Service handles decisions about the content of the system's dialogue acts (what to say) whereas the Scheduler manages the turn-taking aspect of the dialogue (when to take the floor). This separation is also an advantage when using such a multi-layer architecture; turn-taking strategies can be designed independently and embedded in the Scheduler. In the following, this approach is used in order to analyse the effect of several turn-taking strategies.

4. Incremental dialogue simulation

In this section, a simulated environment with a User Simulator, a Scheduler and a Service is introduced. The environment presentation is initiated with the task at hand: an assistant for personal agenda. Yet all notions can be easily generalised to other comparable slot-filling goal-directed dialogue tasks. Several slot-filling strategies have been implemented in the Service as well as several turn-taking phenomena (TTP). Then, experiments are run using a few scenarios in order to analyse the impact of these strategies in terms of dialogue duration and task completion ratio.

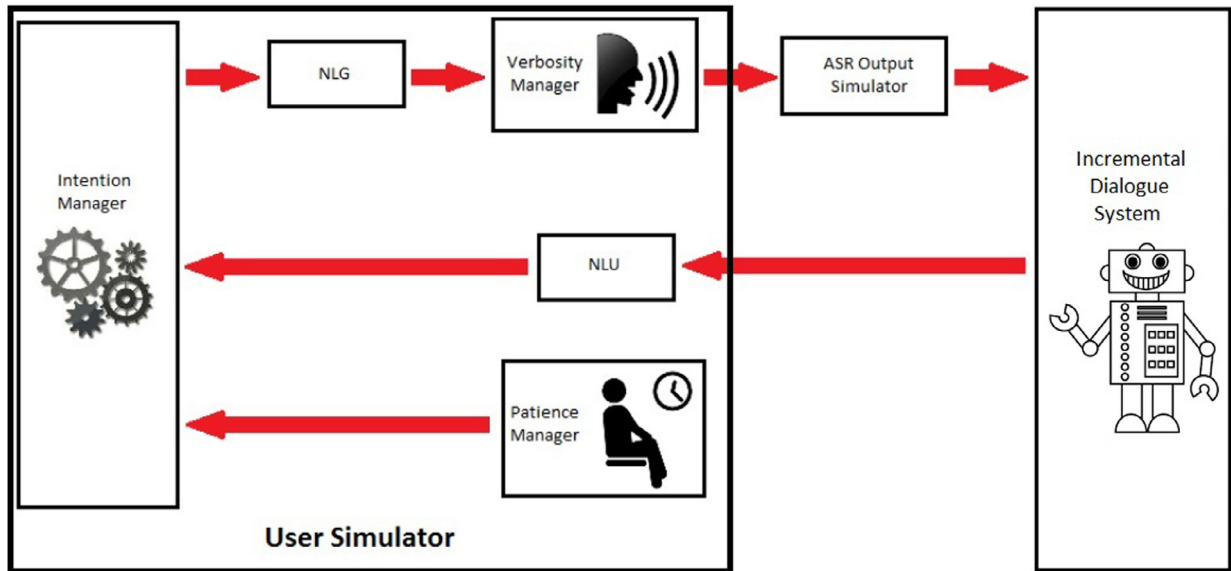


Fig. 4. Simulated environment architecture.

4.1. Simulation environment

Fig. 4 presents the architecture of the simulated environment introduced here. It is composed of three main parts: the User Simulator, the Scheduler and the Service. In the following, the Service is presented first in order to introduce the task used for the experiments, then the User Simulator is described in detail.

4.1.1. The Service

The developed Service aims to help the user manage her personal agenda.⁵ It is fully functional and able to interact with humans (see Section 6). A user can perform three kinds of actions: add, modify or delete an event. An event is defined by three slots: its description (*football game* or *house cleaning* for example), its date and its time window. For instance, a user request may be: *I would like to add the event 'football game' on March 3rd from 9pm to 10pm, Move the event 'house cleaning' to November 10th from 2pm to 4pm* or *Please delete the event 'call John'*. A Service constraint is to have no overlap between events and the description must be a unique event identifier (duplications not allowed).

An NLU module is in charge of parsing the user's request. It uses a simple keyword spotting algorithm coupled with a set of handcrafted rules.

Collecting the slot values can be done using different strategies. Three slot-filling strategies are initially implemented in the Service: System Initiative, User Initiative and Mixed Initiative. Later on, turn-taking strategies are built on top of them in the Scheduler. In the System Initiative (SysIni), the system asks the user for the slot values one by one. If it is unable to understand a user's response, it keeps asking for the same slot until it succeeds to get this information (if, in the meantime, the user does not hang up). The following dialogue illustrates the functioning of this strategy:

```

SYSTEM: What kind of action do you want to perform?
USER: Add.
SYSTEM: Please specify a description.
USER: Birthday party.
SYSTEM: Please give the date of the event.
USER: Jane [noise] 6 [noise].
SYSTEM: Sorry, I don't understand.

```

⁵ The system developed using Orange's Disserto solution works in French, yet an English version is presented here to insure language coherence.

USER: I said, January 6th.
 SYSTEM: Please specify a time window.
 USER: From 9pm to 11pm.
 SYSTEM: Ok. So you want to add the event birthday party on January 6th from 9pm to 11pm.
 USER: Yes.

The User Initiative strategy (UsrIni) asks the user to give all the slot values at once (in one sentence using natural language). If there is still missing information in the user's utterance, she is asked to repeat it again from the beginning (or to reformulate it). For example:

SYSTEM: Please let me know what do you want to do.
 USER: I want to add the event birthday party on [noise] Jane 6 [noise] from 9pm to 11pm.
 SYSTEM: Sorry, I don't understand.
 USER: I said, I want to add the event birthday party on January 6th from 9pm to 11pm.
 SYSTEM: Ok. So you want to add the event birthday party on January 6th from 9pm to 11pm.
 USER: Yes.

Finally, the Mixed Initiative (MixIni) strategy starts by asking the user to formulate a complete request like in the UsrIni case, but if there is still missing information, it switches to SysIni to gather it. For example:

SYSTEM: Please let me know what do you want to do.
 USER: I want to add the event birthday party on [noise] Jane 6 [noise] from 9pm to 11pm.
 SYSTEM: Please give the date of the event.
 USER: I said, January 6th.
 SYSTEM: Ok. So you want to add the event birthday party on January 6th from 9pm to 11pm.
 USER: Yes.

4.1.2. The User Simulator

The User Simulator is composed of five modules: the Intent Manager, the NLU, the NLG, the Verbosity Manager and the Patience Manager. The parameter values chosen in this section have been empirically set to reasonable value, nevertheless, the framework presented here is generic and these values can be modified in order to test several dialogue configurations.

Intent Manager: Following an agenda-based scheme (Wei and Rudnicky, 1999; Schatzmann et al., 2007), the Intent Manager is in charge of computing the User Simulator's dialogue acts. At each dialogue it is provided with a list of events that already exist in the agenda as well as the list of events that it is supposed to add to the agenda during the interaction. These events are associated with a priority value and in the case of an overlap, the Intent Manager decides to move or even delete some of them in order to keep the ones with the highest priority.

NLG and Verbosity Manager: The NLG module transforms the Intent Manager's output into a straightforward request like *Add the event birthday party on January 6th from 8pm to 11pm*. It also generates responses to more targeted questions (about a specific slot), for example *March 3rd* or *from 8am to 9am*.

The Verbosity Manager adds prefixes and suffixes to the output of the NLG in order to add variability in a realistic fashion. A corpus study led in Ghigi et al. (2014) shows that when communicating with a vocal Service, the users tend to repeat the same information several times especially when the system does not understand them the first time. The Verbosity Manager also replicates this kind of behaviour for example: *January 6th, I said January 6th*, with a probability of 0.3 after a misunderstanding is declared. Ghigi et al. (2014) also report that one request over ten is out of domain which is also replicated by the Verbosity Manager (random sentences are uttered with a probability of 0.1).

Patience Manager: While interacting with dialogue systems, users might hang up before the end of the dialogue for several reasons: tedious task accomplishment, annoying behaviour, etc. To model this, a Patience Manager makes the User Simulator hang up when the dialogue lasts for more than $t_{pat} = \frac{2\mu_{pat}}{1+e^{-X}}$ with $X \sim \mathcal{N}(0, 1)$ and $\mu_{pat} = 3$ min. To estimate the time elapsed since the beginning of the dialogue, a speech rate of 200 words per minute is assumed (Yuan et al., 2006) in order to compute the dialogue duration directly from the number of words pronounced both by the user and the system. Moreover a silence of one second is assumed during system to user transitions and two seconds the other way around. In the case of a barge-in or an accurate end point detection, no silence comes into play.

NLU: The NLU module gets the system's response as an input and returns a dialogue act. The set of possible dialogue acts is known in advance, therefore, this module only performs a straightforward mapping with the exception of the confirmation dialogue act which comes with a set of parameters (the slots to confirm).

4.2. ASR Output Manager

In the case of traditional dialogue systems, ASR is requested at the end of the user's utterance (generally marked by a long enough silence). It outputs an N -Best list, which corresponds to the top N recognition hypotheses with their associated confidence scores. To perform incremental dialogue processing, the ASR is solicited as the user speaks (at a fixed frequency for example). The main challenge raised by this mechanism is *ASR instability*: the output at time t_1 is not necessarily a prefix of the output at time t_2 .

To our knowledge, the only incremental ASR simulator is the one introduced in [Selfridge and Heeman \(2012\)](#). It is implemented in such a way that ASR instability is not taken into account to keep things simple. In this new simulator, a new mechanism to simulate this phenomenon is proposed. At each micro-turn, the User Simulator pronounces one word (one and only one word is sent to the ASR Output Simulator). The ASR Output Simulator computes a new N -Best corresponding to the new word only: *the word N-Best*. Let $scramble(w)$ be a word that is different from w and that is taken randomly from a dictionary with probability 0.7, an empty string with probability 0.15 or w concatenated (adding a space between the two) with a random additional different word with probability 0.15. Given a specific Word Error Rate (WER , here controlling the noise level), the word N -Best is computed as follows (the new word is called w_{t+1}):

1. Determine whether w_{t+1} is among the word N -Best or not. An additional parameter is involved in this decision: In N -Best Factor ($INBF$). w_{t+1} is declared to be among the word N -Best with a probability of $(1 - WER) + INBF * WER$. If it is not the case, the word N -Best is totally made of $scramble(w_{t+1})$ samplings and go to step 4.
2. With $(1 - WER)$ probability, w_{t+1} is the best hypothesis of the word N -Best, otherwise its position is uniformly chosen between 2 and N .
3. All other positions are filled with samplings from $scramble(w_{t+1})$.
4. The score associated with the first hypothesis is sampled as $\sigma(X)$ where $X \sim \mathcal{N}(1, 1)$ if this hypothesis is correct or $X \sim \mathcal{N}(-1, 1)$ otherwise. This leads to a distribution with a mean of 0.7 for the correct case and 0.3 for the incorrect one. Both have a standard deviation of 0.18.
5. The scores associated with hypotheses 2 to N are calculated in an iterative fashion: let s_i be the score of the i th hypothesis, s_i is uniformly sampled in $[0, s_{i-1}]$.

Once the new word N -Best is computed, it is integrated to the current utterance N -Best. Let $(s_i)_{1 \leq i \leq N}$ be the scores from the current utterance N -Best and $(s'_i)_{1 \leq i \leq N}$ be the scores from the new word N -Best. The score associated with the combination of the i th current utterance hypothesis and the j th new word hypothesis is considered to be $s_i s'_j$. First, the combination scores are computed then the scores corresponding to combinations that give birth to a new NLU input (the identification of a new information slot) are boosted as follows:

$$s_i \leftarrow s_i + BF(1 - s_i)$$

where BF (Boost Factor) is parameter in $[0, 1]$. This mechanism enables to simulate the ASR instability.

4.3. TTP implementation

A new taxonomy of turn-taking phenomena (TTP) has been recently introduced ([Khouzaimi et al., 2015b](#)) and five of them have been identified as being the most likely to improve the dialogue efficiency: `FAIL_RAW`, `INCOHERENCE_INTERP`, `FEEDBACK_RAW` and `BARGE_IN_RESP` both from the user's and from the system's perspective. In this section, a brief definition of these TTP is provided as well as a description of the way they are implemented in the Scheduler.

The Scheduler is able to perform three different actions: `WAIT`, `SPEAK` and `REPEAT`. The first one designates the decision not to retrieve the last Service's response to the Client whereas the second corresponds to the case when the Scheduler decides to commit to the last user's request thus communicating the response to the Client which in turn will take the floor by relaying it to the TTS (this decision should be based on the last stable utterance since the last utterance is likely to change). When `REPEAT` is chosen, the Scheduler sends the last word of the last stable utterance to the TTS.

Before providing the implementation details of the different TTP, it is important to introduce some new concepts regarding the ASR instability. As explained earlier, the current partial utterance is not necessarily a prefix of the ones coming later. However, words that came at an early stage of the utterance and were not modified since then are more likely to stay unchanged than the ones that have just been pronounced. In McGraw and Gruenstein (2012), it is shown that the part of a partial utterance that has been pronounced more than 0.6 seconds earlier has 90% percent chance of staying unmodified. As the speech rate is supposed to be 200 words per minute in this work, this corresponds to a duration of two words which is called the *Stability Margin* ($SM = 2$). The current partial utterance without this margin is called the *last stable utterance*.

The TTP taxonomy of Khouzaimi et al. (2015b) describes human conversation but in the following, we directly apply it to human–machine dialogue. The definition as well as the rules behind the implementation of each one of the selected TTP are described in the following. Prior to any experiment, several parameter combinations have been tried and among the ones leading to realistic behaviours, we picked the best one.

FAIL_RAW: It corresponds to the situations where the user holds the floor for long enough while failing to deliver his message (due to noise for example) hence being interrupted by the system in order to report the problem and ask him to repeat or reformulate. Concerning our system, when the user speaks for long enough without providing any information that can be parsed by the NLU, the Scheduler decides to SPEAK. A number-of-word threshold is set depending on the last system’s question (the kind of information it is waiting for since some are longer than others): 6 words for open questions, 3 for yes/no questions, 4 for dates and 6 for time windows.

INCOHERENCE_INTERP: Even though the system perfectly understands a user’s partial request, it can still have a good reason to interrupt the user before he finishes speaking. This happens when the content of the request is problematic or in conflict with the current dialogue context. For example, when the user says *I want to move the event football game to...* and there is no football game event already present in the agenda, interrupting the user to report the problem makes sense as it can save him time and energy. Here, the Scheduler decides to SPEAK as soon as the current stable utterance results in a conflict with an existing event or when it refers to a non-existing one (like in the previous example).

FEEDBACK_RAW: If a word or an expression is pronounced by the user, the system can repeat it without interrupting the user. For example:

USER:	01 45...
SYSTEM:	01 45
USER:	25
SYSTEM:	35
USER:	No, 25
SYSTEM:	Sorry, 25
USER:	12 58
SYSTEM:	12 58

To keep things simple, the system repeats the last word of the last stable utterance only. It does so when the score of this word drops below a certain threshold. The user simulator generates each utterance word by word (one word corresponds to one micro-turn). Each new word is associated with a local score called the *word score* (Khouzaimi et al., 2016). The score associated with the whole partial utterance (which we simply call *score* here) is the product of all the word scores of all the words composing it (as well as empty words in case of a deletion ASR error). Therefore, the score associated with the last word is the ratio s_{t-SM}/s_{t-SM-1} (recall that the available ASR scores are related to the whole partial utterance), then the Scheduler decides to REPEAT when this ratio is below 0.7.

BARGE_IN_RESP (System): When the user already provided all the necessary information and the latter is likely to be stable, barging-in right away might save time and energy to the user. It can also prevent him from providing misleading information to the system (Ghigi et al., 2014). Therefore, as soon as the last stable utterance is considered as being enough to provide an answer, the Scheduler decides to SPEAK.

BARGE_IN_RESP (User): Symmetrically, the user can also barge-in if she thinks that she understood the system’s message before its end (happens when the user gets familiar with the system). No Scheduler decision is involved in this TTP since only the user can choose to barge-in or not (in real dialogues, this happens when the user gets familiar with the system). Therefore, an option has been implemented in the User Simulator and when it is activated, the latter takes the floor before the end of some systems responses.

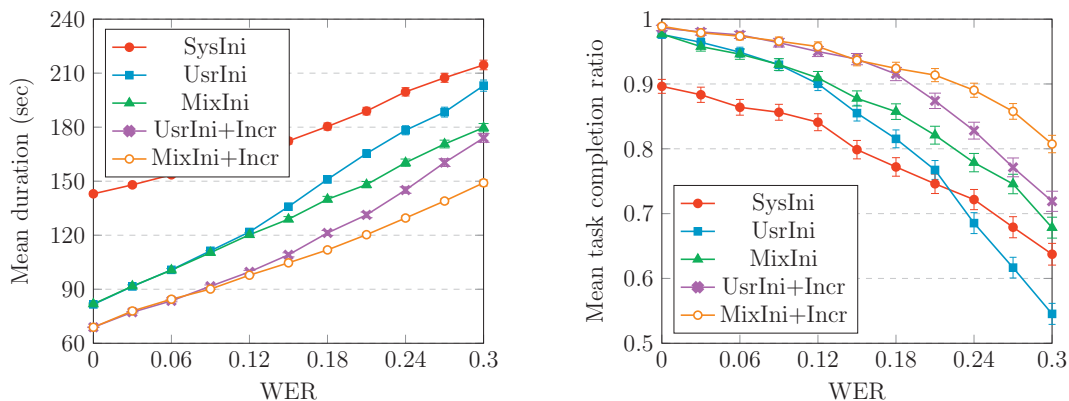


Fig. 5. Mean dialogue duration and task completion for slot-filling strategies combined with the new handcrafted turn-taking strategy.

4.4. Experiment and results

Dialogue efficiency is measured using two objective metrics⁶: the dialogue duration and the task completion ratio. In the following, each strategy is evaluated over 3 scenarios with 1000 dialogues for each *WER* level. The *WER* varies between 0 and 0.3 with a step of 0.03.

First the three slot-filling strategies implemented are compared: SysIni, UsrIni and MixIni. The results are depicted in Fig. 5 (the means over the three scenarios with 1000 dialogue each and the 95% confidence intervals). SysIni is more tedious than UsrIni for the user but above a *WER* = 0.24 it becomes significantly better in terms of task completion. This is not visible in the duration graph since the UsrIni distribution is centered on short dialogues with a thick tail, whereas SysIni is centered on long dialogues. Still, MixIni is the best strategy in both low and high noise situations.

Then, incremental strategies have also been implemented: UsrIni+Incr and MixIni+Incr, which correspond to UsrIni and MixIni enhanced with the TTP presented above (implemented in the Scheduler). The utterances used in SysIni are very short, therefore, SysIni+Incr has not been implemented. Incremental behaviour significantly improves UsrIni performances and it even achieves better results than MixIni. Finally, MixIni+Incr provides the best performance.

5. Optimising turn-taking with RL

So far, all the incremental strategies tested relied on handcrafted rules. In this section, RL is used to automatically learn an optimal strategy while interacting with the User Simulator. The agent which actions are to be optimised is the Scheduler. At each micro-turn, its state is updated and it has to perform one of the two following actions: WAIT or SPEAK. REPEAT is more complex to optimise using RL and it requires more modelisation effort, thus, it is left for future work.

5.1. RL: background

RL is a machine learning framework where an agent tries to learn an optimal behaviour (called *optimal policy*) in order to maximise a *reward function* in an environment (Sutton and Barto, 1998). It learns under the trial-and-error paradigm. This agent is usually modelled as a Markov Decision Process (MDP) which is a quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ where

- \mathcal{S} is the state space. The state at time t is called s_t .

⁶ Spoken dialogue system evaluation is a research thread in itself and many frameworks have been developed to tackle this problem, both in academia (Walker et al., 1997; Hone and Graham, 2000; Schmitt et al., 2012) and industry (Evanini et al., 2008; Witt, 2011). In simulation though, only objective metrics can be measured. Here, we focus on the two mainly used metrics.

- \mathcal{A} is the action space. The action at time t is called a_t .
- \mathcal{T} is the transition model. In other words, it is the set of probabilities $\mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$ for all $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$.
- \mathcal{R} is the reward model. After choosing action a_t in state s_t , the agent receives a stochastic reward $r_t \in \mathbb{R}$. \mathcal{R} is the reward density $\mathbb{P}(r_t = r | s_t = s, a_t = a)$ for each $(s, a, r) \in \mathcal{S} \times \mathcal{A} \times \mathbb{R}$. Along with \mathcal{T} , they define the dynamics of the MDP.
- $\gamma \in [0, 1)$ is the discount factor, used to weight future rewards.

At time t , the *return* is defined as $R_t = \sum_{k \in \mathbb{N}} \gamma^k r_{t+k}$. A deterministic *policy* π is mapping between states and actions ($\pi : \mathcal{S} \rightarrow \mathcal{A}$). Following that policy means taking action $\pi(s)$ whenever the agent is at state s . The expected return for being at state s and following π afterwards defines a function V over \mathcal{S} as follows:

$$V^\pi(s) = \mathbb{E}_\pi[R_t | s_t = s] \quad (1)$$

The objective of RL is to learn an optimal policy π^* , such that $\forall \pi \in \mathcal{A}^{\mathcal{S}}, \forall s \in \mathcal{S}, V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$.

RL algorithms also use the expectation for being at state s , taking action a and following the policy π afterwards, called the Q -function:

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \quad (2)$$

The Q -function corresponding to the optimal policy is called $Q^* = Q^{\pi^*}$, and verifies the *Bellman optimality equation*:

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')] \quad (3)$$

with r being the stochastic reward after taking action a while being at state s , and s' being the stochastic next state. This property is used by many algorithms to approximate Q^* , like it is the case for the Fitted- Q algorithm introduced in [Section 5.4](#).

5.2. State representation

At each micro-turn, the Scheduler's state is determined by the following features:

- **SYSTEM_REQ**: The information slot that the Service is currently waiting for. It can be a specific slot, all the slots at once or a response to a yes/no question. This variable can take 6 different values.
- **LAST_INCR_RESP**: The last response that has been returned by the Service (as discussed in [Section 3](#)). There are 11 different alternatives.
- **NB_USER_WORDS**: The number of words in the current partial utterance since the last LAST_INCR_RESP change.
- **NORMALISED_SCORE**: Since confidence scores are between 0 and 1 and the boosting is performed only when new concepts appear, the longer the sentence, the more likely it is to have a low score (even if the scores associated with each word N -Best are high). Therefore, instead of taking the raw score as a feature, we consider a normalised version by taking the geometrical mean with respect to the number of words n : $\sqrt[n]{score}$.
- **TIME**: Total duration of the dialogue until the current micro-turn.

The objective is to use these variables to build a linear model representing the Q -function (other types of models are possible but this one is chosen for its simplicity). Therefore, they are used to form new variables that are more adapted to such model. Only 21 combinations of the two first features are frequent and the others happen rarely. Since categorical variables cannot be directly used in a linear model, these two features are used to form 21 new Kronecker variables $\delta_1, \dots, \delta_{21}$. The variable δ_i equals 1 when the i th combination occurs and 0 otherwise.

NB_USER_WORDS cannot be directly incorporated into a linear model of the Q -function neither. By doing so, monotony is automatically assumed which is not a desired effect. Therefore, three Radial Basis Functions (RBFs) are used as a proxy for this information: ϕ_1^{nw} , ϕ_2^{nw} and ϕ_3^{nw} . They are defined as follows:

$$\phi_i^{nw} = \exp\left(\frac{(NB_USER_WORDS - \mu_i)^2}{2\sigma_i^2}\right)$$

$$\mu_1 = 0, \mu_2 = 5, \mu_3 = 10$$

$$\sigma_1 = 2, \sigma_2 = 3, \sigma_3 = 3$$

NORMALISED_SCORE is also represented in a similar way using two RBFs: ϕ_1^{ns} and ϕ_2^{ns} . They are centered in 0.25 and 0.75 respectively with a standard deviation of 0.3 for both.

The *TIME* variable is normalised as follows:

$$T = \text{sigmoid}\left(\frac{TIME - 180}{60}\right)$$

Finally, at each micro-turn, the dialogue state s is represented by the following vector:

$$\Phi(s) = [1, \delta_1, \dots, \delta_{21}, \phi_1^{nw}, \phi_2^{nw}, \phi_3^{nw}, \phi_1^{ns}, \phi_2^{ns}, T] \quad (4)$$

5.3. Episodes and rewards

A learning episode is a portion of dialogue where the user performs a single operation on the agenda (adding modifying or deleting an event).

As far as rewards are concerned, at each micro-turn, the Scheduler receives $-\Delta t$ which corresponds to the time elapsed since the last decision (during the last micro-turn). Moreover, when the task is completed, the Scheduler receives a reward of 150.

5.4. Fitted-Q Value Iteration

As shown in [Section 5.1](#), in Reinforcement Learning, the Q -function Q^* associated with the optimal policy verifies the Bellman optimality equation, which can also be written in the following form:

$$Q^* = T^*Q^* \quad (5)$$

The Bellman optimality operator T^* can be shown to be a contraction and according to the Banach theorem it admits a unique fixed point. The latter can be found by value iteration, nevertheless, an exact representation of Q^* is needed which is rarely the case, and certainly not in our application. As a consequence, a linear approximation \hat{Q}^* is computed instead. By denoting P the projection operator on the space of all the linear approximations, instead of solving [Eq. \(5\)](#), we solve the following one:

$$\hat{Q}^* = PT^*\hat{Q}^* \quad (6)$$

For that purpose, Fitted- Q Value Iteration ([Richard Bellman, 1959](#); [Samuel, 1959](#); [Chandramohan et al., 2010](#); [Szepesvari, 2010](#)) has been used (we suppose that PT^* is still a contraction which is generally the case in practice). It is a batch RL algorithm where our Q -function is linearly approximated as follows

$$\hat{Q}(s, a) = \theta(a)^T \Phi(s) \quad (7)$$

where $\theta(a)$ is a parameter vector for action a and $\Phi(s)$ is the state representation vector defined in [Eq. \(4\)](#). The algorithm takes a set of transitions (s_t, a_t, r_t, s_{t+1}) (the batch) and outputs the $\theta(a)$ vector associated with each action that represents the Q -function associated with the learnt policy (the policy that always takes the greedy action with respect to Q).

The associated θ vectors are iteratively computed as follows (with $\gamma = 0.99$ and $\mathcal{I}(a)$ the indexes of the elements in the batch where action a has been taken):

$$\theta_i(a) = \arg \min_{\theta} \sum_{j \in \mathcal{I}(a)} \left(r_j + \gamma \max_a \theta_{i-1}(a')^T \phi(s'_j) - \theta(a_j) \phi(s_j) \right)^2$$

$$= \left(\sum_{j \in \mathcal{I}(a)} \phi(s_j)^T \phi(s_j) \right)^{-1} \sum_{j \in \mathcal{I}(a)} \phi(s_j) \left(r_j + \gamma \max_a \theta_{i-1}(a')^T \phi(s'_j) \right)$$

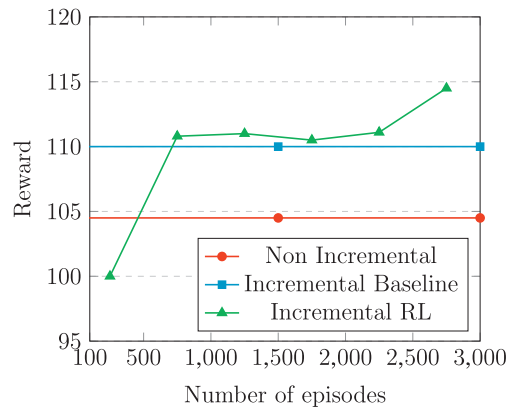


Fig. 6. Learning curve (0–500: pure exploration, 500–2500: exploration/exploitation, 2500–3000: pure exploitation). $WER = 0.15$.

5.5. Experiment and results

Fitted- Q is used here in an online manner as every 500 episodes, it is run to update the model based on all the transitions that happened from the beginning of the process. Initially, the θ vectors are set to zero. The Scheduler learns through three stages:

1. *Pure exploration (Episodes 0–500)*: At each micro-turn, the action WAIT is randomly chosen with a probability of 0.9 and the action SPEAK is performed with a probability of 0.1.
2. *Exploitation/exploration (Episodes 500–2500)*: An ϵ -greedy ($\epsilon = 0.1$) with respect to the learnt Q -function is used.
3. *Pure exploitation (Episodes 2500–3000)*: The Scheduler is totally greedy with respect to the current Q -function (the learned policy is tested).

The learnt strategy is compared to two different baselines: the non-incremental strategy and a slightly modified version of the the MixIni+Incr which is the best handcrafted rule-based strategy presented in Section 4. During the learning process, the WER is fixed at 0.15. Fig. 6 illustrates this process with the three learning stages (recall that the reward is computed for each episode as $150TC\text{-duration}$ where TC equals 1 in case of task completion and 0 otherwise). During the pure exploration phase, the Scheduler performs worse than both baselines, then during the exploitation/exploration it shows similar performances as the incremental baseline. Finally, in the pure exploitation phase, it achieves even better performances hence being the best strategy introduced in this paper.

Fig. 7 compares the two baselines and the new learnt strategy in terms of dialogue duration and task completion under different noise conditions (learning has been performed for $WER = 0.15$, but the learnt strategy is tested under other WER values). For $WER = 0.3$, the handcrafted strategy finishes 10 seconds earlier than the non-incremental one and the RL strategy still makes it possible to gain 20 additional seconds (total gain of 17%). As far as task completion is concerned, at the same noise level, the non-incremental and the handcrafted baselines respectively achieve a score of 70 and 73%, whereas the automatically learnt strategy keeps this ratio above 80%.

6. Live experiment

So far, all the results presented were obtained in simulation only. In this section, a live experiment with real users is presented in order to evaluate our turn-taking strategies in real dialogue conditions.

The approach presented here has been transposed to a new domain to run a live experiment with real users: the Majordomo (the handcrafted strategy is transposed from the previous one).

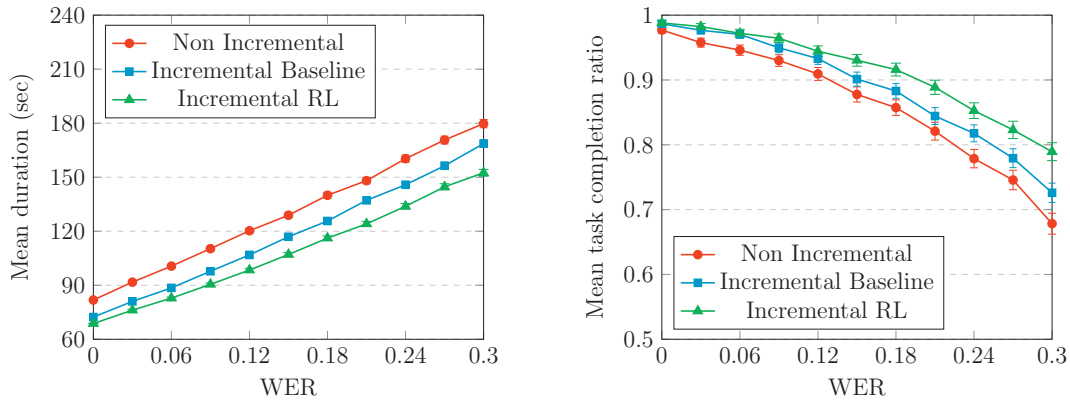


Fig. 7. The non-incremental, the handcrafted and the RL strategies compared under different levels of noise.

6.1. The Majordomo domain

The Majordomo spoken dialogue application has been developed as part of the FUI project VoiceHome at Orange Labs. This shows the generality across domains of the approach developed in this paper. The users were given specific tasks to accomplish while interacting with a smart home. They could schedule domestic tasks, modify already scheduled ones or cancel them. Some tasks cannot be run simultaneously, like *air conditioning* and *heating*, or *mowing lawn* and *watering lawn* for example. The different possible tasks are depicted in the left column of Table 2 and for each one, the conflicting tasks (tasks that cannot be scheduled at the same time as the main task) are listed in the right column.

6.2. Implementation

The Client is implemented as a web page using JavaScript. Google ASR has been used since it offers state-of-the-art performances with the possibility to output incremental results. To estimate the WER, 500 user utterances have been processed and 228 errors have been detected with a total of 2497 words. Therefore, the estimated WER for this experiment is $9\% \pm 1\%$ ($\alpha = 0.05$). For speech synthesis, Google’s solution has also been maintained.

The Service has been implemented using the Disserto solution at Orange Labs and a slightly modified version of the Scheduler presented before has been used in order to fit the new domain and the real time constraint (the number

Table 2
Majordomo tasks.

Task	Conflicting tasks
Alarm	Laundry, hoover
Heating	Open windows, air conditioning
Open windows	Heating, air conditioning
Absence mode	Open windows
Laundry	Alarm, calm mode
Air conditioning	Heating, open windows
Swimming pool warming	Swimming pool cleaning
Swimming pool cleaning	Swimming pool warming
Calm mode	Laundry, hoover, mow lawn
Mow lawn	Water lawn
Water lawn	Mow lawn
Record channel 1	Record channel 2, record channel 3
Record channel 2	Record channel 1, record channel 3
Record channel 3	Record channel 1, record channel 2
Hoover	Alarm, calm mode
Run bath	

of words has been replaced with real time). The handcrafted and the data-driven strategies were built using the same approach as in the agenda domain. The RL strategy was trained in simulation before being tested with real users in pure exploitation mode. Online learning with real users while evaluating the strategy at the same time requires a bigger data collection and it is therefore left for future work.

6.3. Experimental protocol

Since the Client is a web page, users were able to remotely interact with the Majordomo using their own devices. They started the experiment by reading the following instructions in a welcome page before being redirected to the dialogue interface:

- In order to collect enough data, we ask our participants to perform 5 dialogues with the system.
- All the dialogues should be performed using the same computer and under the same conditions.
- At the end of each dialogue, you will be asked to answer a few questions about the last interaction. Please fill in one form per dialogue.
- If you get stuck in a dialogue or you think that the interaction is too tedious or takes too long, you can end it by saying “goodbye” or by clicking on the red “Hang up” button.

The dialogue scenario is randomly chosen among 10 existing ones. The turn-taking strategy is hidden from the user and randomly picked among the non-incremental, the handcrafted incremental and the RL one.

At the end of the dialogue, each user filled a survey where the following Key Performance Indicators (KPIs) were collected:

- *Reactivity*: Is the system reactive? (1–6).
- *Reactivity quality*: Does the system’s reactivity improve the dialogue quality? (1–6).
- *Human-likeness*: To what extent does the system behave like a human being? (1–6).
- *Efficiency*: How would you rate the system’s efficiency? (1–6).
- *Global quality*: How would you rate the global quality of the dialogue? (1–6).
- *Potential use*: If Majordomo was an available product on the market, would you use it at home? (1–4).

6.4. Results

47 participants performed 206 dialogues: 65 for the non-incremental strategy (*None*), 65 for the handcrafted incremental one (*Handcrafted*) and 76 using the RL one (*RL*).

Both the objective and subjective metrics results are depicted in [Table 3](#). It appears that the incremental strategies reduce the dialogue duration even though this is not statistically significant. Similarly, in [Ghigi et al. \(2014\)](#), the proposed incremental strategy increases the dialogue duration since users were often interrupted before they provided all the information they wanted to communicate. However, no difference exists between these two strategies in terms of dialogue duration.

Table 3
Global dialogue evaluation metrics.

Category	KPI	<i>None</i>	<i>Handcrafted</i>	<i>RL</i>
Objective	Duration (s)	94.7	89.6	90.6
	Task completion	60%	63%	75%
	Reactivity	4.31	4.57	4.62
	Reactivity quality	4.38	4.25	4.36
Subjective	Human-likeness	3.63	3.66	3.74
	Efficiency	4.22	4.20	4.36
	Global quality	4.06	4.18	4.20
	Potential use	2.66	2.68	2.82

Table 4
Global dialogue evaluation metrics.

KPI	<i>None</i>	<i>Handcrafted</i>	<i>RL</i>
Latency (ms)	1545 ± 61	1303 ± 78	588 ± 59
FC ratio	No barge-in	0.31 ± 0.091	0.068 ± 0.023

The significant⁷ added value of *RL* turns out to be the task completion ratio. It improves it by 15% compared to *None* ($p = 0.030$). Compared to *Handcrafted*, the improvement is still visible (12%, $p = 0.065$). The Majordomo task involves a certain cognitive load and the users must be focused in order to be able to perform the different subtasks in the right order while keeping the final objective in mind. When interacting with the non-incremental version of the Majordomo, users showed less engaged behaviours compared to interactions using *RL*. The *RL* version of the Majordomo is perceived as being more proactive which helped the users to stay focused on the task whereas the *None* version is perceived as being more passive. As far as *Handcrafted* is concerned, even though it is an incremental and more reactive strategy, it does not produce the same effect as *RL* which shows that it poorly manages turn-taking decisions.

To support that, a turn-taking local study has been performed: the average system’s response latency has been computed (549 transitions for *None*, 542 for *Handcrafted* and 727 for *RL*), moreover, for *Handcrafted* and *RL*, all the SPEAK decisions (*Handcrafted*: 99, *RL*: 456) have been manually annotated for being a good (the system was right to take the floor) or a bad decision (the system should have waited longer before taking the floor) in order to compute the false cut-in (FC) ratio (Raux and Eskenazi, 2012). The corresponding results are given in Table 4 ($p < 0.000001$). It appears that *Handcrafted* does not take much risk since it rarely decides to SPEAK compared to *RL* and when it does, it is poorly managed since it is a bad decision one third of the time. *RL* on the other hand decides to spontaneously take the floor more often and when it does, it is a good decision most of the time (FC ratio of 6.8% only).

As far as subjective metrics are concerned, their variances are much higher. Therefore, apart from the *Reactivity* KPI that is significantly improved by *RL* compared to *None* ($p = 0.048$), the other differences are not statistically significant. Nevertheless, the general tendency is in favour of *RL*.

6.5. Complementarity with acoustic clues

After the user trials we wanted to confirm that our proposed approach can improve the performance when combined with audio features and looked for analysis results to support this assumption. In that purpose, a detailed analysis of the live experiment data has been carefully handled by speech experts. For each SPEAK decision, the experts were asked whether there are acoustic clues that could have triggered the Scheduler to take the floor or not. We have shown that 24% of these decisions could not have been triggered with pure acoustic features only (453 good turn decisions proposed by the system have acoustic clues available, mainly pause marks and varying intonation, and 147 can only be attributed to content analysis, out of a total of 676 analyzed speech turns). It has also been observed that in the vast majority of the first set of decisions (where acoustic and semantics clues simultaneously exist) the system based on content features is systematically a few milliseconds ahead of the acoustic evidences. Many observations correspond to closed questions for which the system interrupts the user as soon as it has a consistent answer and thus avoids the remaining filler part provided by the user ; for instance, if the user utters “yes that’s right”, the system can start its next turn right after the “yes” is uttered and thus appears much more reactive (due to processing latency, the TTS may not be produced before the end of the whole sentence anyway, but with less delay). Anyhow, overall this tends to confirm that semantics are complementary to acoustic clues since they represent a new source of relevant information for turn-taking (in this line, our study confirms what has been observed in previous work (Raux and Eskenazi, 2012; Meena et al., 2013)).

⁷ To measure statistical significance, a Welsh *t*-test and a binomial proportions test have been used, with very comparable *p*-values.

7. Conclusion and future work

A new methodology for turn-taking capacities enhancement using RL is introduced in this paper. Starting from a traditional dialogue system, a new approach for transforming it into an incremental one is presented. A first simulation study made it possible to compare a non-incremental strategy, a rule-based turn-taking strategy which is the fruit of a human conversation analysis using a turn-taking phenomena taxonomy, and an automatically learnt strategy using RL. The rule-based strategy is shown to improve the non-incremental baseline but the RL strategy is shown to offer the best results. As a validation, an experiment with real users have been run and the results show that RL significantly improved the task completion ratio (15% over the non-incremental strategy) while reducing dialogue duration and slightly improving the users' subjective evaluation of the system.

For future work, we plan to explore the ability of the RL strategy to learn directly from real interactions instead of simulation. Moreover, the collected dialogues will be used to better calibrate the simulated environment's parameters. In addition, methods to learn how to perform the REPEAT action efficiently will be explored as well as the implementation of other TTP that have not been selected here. Finally, acoustic clues will be integrated in the turn-taking decision process as a complementary source of information.

References

- Aist, G., Allen, J., Campana, E., Gallo, C.G., Stoness, S., Swift, M., Tanenhaus, M.K., 2007. Incremental understanding in human–computer dialogue and experimental evidence for advantages over nonincremental methods. In: *Proceedings of the Eleventh Workshop on the Semantics and Pragmatics of Dialogue (SemDial/Decalog)*.
- Allen, J., Ferguson, G., Stent, A., 2001. An architecture for more realistic conversational systems. In: *Proceedings of the Sixth International Conference on Intelligent User Interfaces (IUI)*.
- Asri, L.E., 2016. *Learning the Parameters of Reinforcement Learning from Data for Adaptive Spoken Dialogue Systems*. Orange Labs/University of Lille (Ph.D. thesis).
- Chandramohan, S., Geist, M., Pietquin, O., 2010. Optimizing spoken dialogue management with fitted value iteration. In: *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association (Interspeech)*.
- Dethlefs, N., Hastie, H.W., Rieser, V., Lemon, O., 2012. Optimising incremental dialogue decisions using information density for interactive systems. In: *Proceedings of the Seventeenth Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.
- Dohsaka, K., Shimazu, A., 1997. A system architecture for spoken utterance production in collaborative dialogue. In: *Proceedings of the Workshop on Collaboration, Cooperation and Conflict in Dialogue Systems at IJCAI*.
- Eckert, W., Levin, E., Pieraccini, R., 1997. User modeling for spoken dialogue system evaluation. In: *Proceedings of the First IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.
- El Asri, L., Lemonnier, R., Laroche, R., Pietquin, O., Khouzaimi, H., 2014. NASTIA: negotiating appointment setting interface. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC)*.
- Evanini, K., Hunter, P., Liscombe, J., Suendermann, D., Dayanidhi, K., Pieraccini, R., 2008. Caller experience: a method for evaluating dialog systems and its automatic prediction. In: *Proceedings of the Second IEEE Workshop on Spoken Language Technology (SLT)*.
- Ghigi, F., Eskenazi, M., Torres, M.I., Lee, S., 2014. Incremental dialog processing in a task-oriented dialog. In: *Proceedings of the Fifteenth Annual Conference of the International Speech Communication Association (Interspeech)*.
- Hone, K.S., Graham, R., 2000. Towards a tool for the subjective assessment of speech system interfaces (SASSI). *Nat. Lang. Eng.* 6, 287–303.
- Jonsdottir, G.R., Thorisson, K.R., Nivel, E., 2008. Learning smooth, human-like turntaking in realtime dialogue. In: *Proceedings of the Eight International Conference on Intelligent Virtual Agents (IVA)*. Springer, pp. 162–175.
- Khouzaimi, H., Laroche, R., Lefèvre, F., 2014a. An easy method to make dialogue systems incremental. In: *Proceedings of the Fifteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- Khouzaimi, H., Laroche, R., Lefèvre, F., 2014b. Vers une approche simplifiée pour introduire le caractère incrémental dans les systèmes de dialogue. In: *Proceedings of the TALN 2014 Conference*.
- Khouzaimi, H., Laroche, R., Lefèvre, F., 2015a. Optimising turn-taking strategies with reinforcement learning. In: *Proceedings of the Sixteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- Khouzaimi, H., Laroche, R., Lefèvre, F., 2015b. Turn-taking phenomena in incremental dialogue systems. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Khouzaimi, H., Laroche, R., Lefèvre, F., 2016. Incremental human-machine dialogue simulation. In: *Proceedings of the International Workshop on Spoken Dialogue Systems (IWSDS)*.
- Kilger, A., Finkler, W., 1995. *Incremental Generation for Real-Time Applications*. Technical Report. German Research Center for Artificial Intelligence.
- Kim, S., Banchs, R.E., 2014. Sequential labeling for tracking dynamic dialog states. In: *Proceedings of the Fifteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- Laroche, R., 2010. *Raisonnement sur les incertitudes et apprentissage pour les systèmes de dialogue conventionnels*. Paris VI University (Ph.D. thesis).
- Laroche, R., Genevay, A., 2016. The negotiation dialogue game. In: *Proceedings of the International Workshop on Spoken Dialogue Systems (IWSDS)*.

- Levelt, W.J.M., 1989. *Speaking: From Intention to Articulation*. MIT Press, Cambridge, MA.
- Levin, E., Pieraccini, R., 1997. A stochastic model of computer–human interaction for learning dialogue strategies. In: *Proceedings of the Fifth biennial European Conference on Speech Communication and Technology (Eurospeech)*.
- Lock, K., 1965. Structuring programs for multiprogram time-sharing on-line applications. In: *Proceedings of the Fall Joint Computer Conference (AFIPS)*.
- Lu, D., Nishimoto, T., Minematsu, N., 2011. Decision of response timing for incremental speech recognition with reinforcement learning. In: *Proceedings of the Ninth IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.
- McGraw, I., Gruenstein, A., 2012. Estimating word-stability during incremental speech recognition. In: *Proceedings of the Twelfth Annual Conference of the International Speech Communication Association (Interspeech)*.
- Meena, R., Skantze, G., Gustafson, J., 2013. A data-driven model for timing feedback in a map task dialogue system. In: *Proceedings of the Fourteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- Paek, T., Pieraccini, R., 2008. Automating spoken dialogue management design using machine learning: an industry perspective. *Speech Commun.* 50, 716–729.
- Paetzel, M., Manuvinakurike, R., DeVault, D., 2015. “so, which one is it?” The effect of alternative incremental architectures in a high-performance game-playing agent. In: *Proceedings of the Sixteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- Pieraccini, R., Huerta, J., 2005. Where do we go from here? Research and commercial spoken dialog systems. In: *Proceedings of the Sixth SIGDIAL Workshop on Discourse and Dialogue*.
- Pietquin, O., Hastie, H., 2013. A survey on metrics for the evaluation of user simulations. *Knowl. Eng. Rev.* 28 (1), 59–73.
- Raux, A., Eskenazi, M., 2009. A finite-state turn-taking model for spoken dialog systems. In: *Proceedings of Human Language Technologies: The Eight Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 629–637.
- Raux, A., Eskenazi, M., 2012. Optimizing the turn-taking behavior of task-oriented spoken dialog systems. *ACM Trans. Speech Lang. Process.* 9 (1), 1:1–1:23.
- Richard Bellman, S.D., 1959. Functional approximations and dynamic programming. *Math. Tables Other Aids Comput.* 13, 247–251.
- Sacks, H., Schegloff, E.A., Jefferson, G., 1974. A simplest systematics for the organization of turn-taking for conversation. *Language* 50, 696–735.
- Samuel, A.L., 1959. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*
- Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., Young, S., 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In: *Proceedings of Human Language Technologies: The Sixth Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.
- Schatzmann, J., Weilhammer, K., Stuttle, M., Young, S., 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowl. Eng. Rev.* 21 (1), 97–126.
- Schlangen, D., Skantze, G., 2011. A general, abstract model of incremental dialogue processing. *Dialogue Discourse* 2, 83–111.
- Schmitt, A., Ultes, S., Minker, W., 2012. A parameterized and annotated spoken dialog corpus of the CMU let’s go bus information system. In: *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC)*.
- Selfridge, E.O., Arizmendi, I., Heeman, P.A., Williams, J.D., 2011. Stability and accuracy in incremental speech recognition. In: *Proceedings of the Twelfth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- Selfridge, E.O., Heeman, P.A., 2010. Importance-driven turn-bidding for spoken dialogue systems. In: *Proceedings of the Forty-Eight Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 177–185.
- Selfridge, E.O., Heeman, P.A., 2012. A temporal simulator for developing turn-taking methods for spoken dialogue systems. In: *Proceedings of the Thirteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue*.
- Singh, S.P., Kearns, M.J., Litman, D.J., Walker, M.A., 1999. Reinforcement learning for spoken dialogue systems. In: *Proceedings of the Thirteenth Annual Conference on Neural Information Processing Systems (NIPS)*.
- Skantze, G., Schlangen, D., 2009. Incremental dialogue processing in a micro-domain. In: *Proceedings of the Twelfth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.
- Sutton, R.S., Barto, A.G., 1998. *Reinforcement Learning, An Introduction*. MIT Press, Cambridge, Massachusetts, London, England.
- Szepesvari, C., 2010. *Algorithms for reinforcement learning*. *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool.
- Tanenhaus, M.K., Spivey-Knowlton, M.J., Eberhard, K.M., Sedivy, J.C., 1995. Integration of visual and linguistic information in spoken language comprehension. *Science* 268, 1632–1634.
- Walker, M.A., Litman, D.J., Kamm, C.A., Abella, A., 1997. Paradise: a framework for evaluating spoken dialogue agents. In: *Proceedings of the Eight Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.
- Wei, X., Rudnicky, A.I., 1999. An agenda-based dialog management architecture for spoken language systems. In: *Proceedings of the Second IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.
- Wirén, M., 1992. *Studies in Incremental Natural Language Analysis*. Linköping University, Linköping, Sweden (Ph.D. thesis).
- Witt, S., 2011. A global experience metric for dialog management in spoken dialog systems. In: *Proceedings of Fifteenth Workshop on the Semantics and Pragmatics of Dialogue (SemDial)*.
- Yuan, J., Liberman, M., Cieri, C., 2006. Towards an integrated understanding of speaking rate in conversation. In: *Proceedings of the Seventh Annual Conference of the International Speech Communication Association (Interspeech)*.
- Zhao, T., Black, A.W., Eskenazi, M., 2015. An incremental turn-taking model with active system barge-in for spoken dialog systems. In: *Proceedings of the Sixteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.