Reinforcement Learning Applied to Incremental Dialogue Systems

Hatim Khouzaimi

Orange Labs / University of Avignon

This dissertation is submitted for the degree of
Doctor of Philosophy

October 2015

I would like to dedicate this thesis to my loving parents …

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Hatim Khouzaimi
October 2015

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Table of contents

List of figures **xv**

List of tables **xvii**

1 Introduction **1**

2 State of the art **3**

2.1 Human dialogue 3

2.1.1 Philosophy of language 3

2.1.2 Turn-taking in human dialogue 4

2.2 Spoken dialogue systems and incremental behaviour 4

2.2.1 Existing architectures 11

2.3 Reinforcement Learning 12

2.3.1 Definition 12

2.3.2 Application to dialogue systems 16

2.3.3 Dialogue simulation 17

2.4 Issues and motivation 18

3 Turn-taking taxonomy **21**

3.1 Taxonomy presentation 21

3.2 Discussion 24

4 Architecture **27**

4.1 Description 27

4.1.1 Overview 27

4.1.2 Time sharing 28

4.1.3 The Scheduler 30

4.2 Illustration 32

4.2.1 A textual dialogue system: CFAsT 32

4.2.2 A spoken dialogue system: Dictanum 33
4.3 Discussion 35
4.3.1 Levels of incrementality 35
4.3.2 Enhancing a traditional dialogue system's turn-taking abilities at a low cost 35
4.3.3 Separating dialogue management from floor management 35

5 Incremental dialogue simulation 37
5.1 Agenda management task 37
5.2 The service 38
5.2.1 Natural Language Understanding 38
5.3 User simulator 39
5.3.1 Overview 39
5.3.2 Intent Manager 40
5.3.3 NLG and Verbosity Manager 40
5.3.4 ASR output simulator 42
5.3.5 Timing and patience manager 44

6 Handcrafted strategies for improving dialogue efficiency 47
6.1 User and system initiative 47
6.1.1 Strategies 47
6.1.2 Experiments 49
6.2 Incremental strategies 50
6.2.1 ASR instability 50
6.2.2 TTP implementation 51
6.2.3 Experiments 52

7 Reinforcement learning for turn-taking optimisation 55
7.1 Model 55
7.1.1 State representation 55
7.1.2 Actions, rewards and episodes 57
7.1.3 Fitted-Q Value Iteration 57
7.2 Experiment 58
7.2.1 Setup 58
7.2.2 Results 59

8 Experiment with real users 63

8.1 Protocol 63

8.2 Results 63

9 A new reinforcement learning approach adapted to incremental dialogue 65

9.1 Problem 65

9.2 Proposed approach 65

9.3 Toy problem application 65

9.4 Simulation results 65

10 Conclusion 67

References 69

Appendix A How to install LaTeX 73

Appendix B Installing the CUED class file 77

List of figures

2.1 The dialogue chain 6
2.2 The interaction between the agent and the environment in reinforcement learning 13
2.3 The alternation between evaluation and control 16

4.1 The Scheduler: an interface between the client and the service 28
4.2 Time sharing in traditional and incremental settings 29
4.3 Incremental behaviour with the Scheduler 30
4.4 Double context management: real and simulated 31
4.5 The incremental version of the CFAsT project 32

5.1 Simulated environment architecture 39
5.2 ASR score sampling distribution 43
5.3 An illustration of the incremental ASR output N-Best update 45

6.1 Simulated mean duration (left) and dialogue task completion (right) for different noise levels 49
6.2 Mean dialogue duration and task completion for generic strategies. 52
6.3 Mean dialogue duration and task completion for different turn-taking phenomena. 53
6.4 INCOHERENCE_INTERP evaluated in a more adapted task 54

7.1 Learning curve (0-500: pure exploration, 500-2500: exploration/exploitation, 2500-3000: pure exploitation) 59
7.2 Mean dialogue duration and task for the non-incremental, the baseline incremental and the RL incremental strategies under different noise conditions. . 61

List of tables

3.1 *Taxonomy labels* 21

3.2 *Turn-taking phenomena taxonomy. The rows/columns correspond to the levels of information added by the floor giver/taker.* 22

4.1 Available features for dialogue systems given the way they integrate incrementality 36

5.1 NLU rules types 38

6.1 *The three scenarios used in the simulation* 50

Chapter 1

Introduction

Introduction here.

Chapter 2

State of the art

2.1 Human dialogue

2.1.1 Philosophy of language

If I say *This dog is big*, I utter a few sounds that can be cast as words. How are these words related to the real objects I refer to? How comes that a sequence of sounds can have effects on others? I can give orders to somebody and make them performs the actions I want as I can congratulate or insult someone and have an emotional impact on her. Also, how comes an utterance can also be judged as a complete nonsense or as a true or false assertion? These are a few questions raised in the philosophy of language.

In his book called *How To Do Things With Words* [1], J.L. Austin focuses on the concept of *speech act* which the title of another book [34] by John Searle, who extended this theory of language. Introducing this concepts is aimed towards bringing answers to the previous questions. Saying *My sister just arrived*, one performs a speech act that can be viewed from different points of view. Suppose that someone is listening as this sentence is being uttered and that person does not speak English, then obviously she only hears a sequence of noises which is the physical, low-level nature of the speech act. When considered from that perspective, the latter is referred to as a *locutionary act*. On the other hand, when the focus is the meaning of the utterance and the message the speakers wants to deliver, the speech act is an *illocutionary act*. Finally, saying something to somebody can have psychological effects on that person: congratulating someone on insulting him car be rewarding or hurting, a strong grounded speech can be convincing...etc... These are referred to as *perlocutionary acts*.

To build a dialogue system, the traditional approach is to consider the user's and the system's speech acts as illocutionary and perlocutionary acts. When studying incremental dialogue systems, the *locutionary act* point of view comes at play. In this thesis, this distinction will be clarified.

2.1.2 Turn-taking in human dialogue

Turn-taking is a sociological phenomenon that is encountered in many different situations: card games, road traffic regulation, CPU resource sharing...etc... In [30], Harvey Sacks describes the social organisation of turn-taking as an economy where *turns are valued, sought or avoided*, depending on the situation at hand. Then, in the rest of his paper, he focuses on the case of human conversation. Obviously, this is subject to many contextual and cultural variations but the objectif of [30] is to meet the challenge of extracting a set of rules that would ultimately describe the human turn-taking mechanisms in a general fashion.

For about six years, Sacks has been analysing conversation recordings and he came up with a few rules that caracterise human conversation and turn-taking. One of us is particularly interesting given the purpose of this thesis: *transition (from one turn to a next) with no gap an no overlap are common. Together with transitions characterized by slight gap or slight overlap, they make up the vast majority of transitions*.

In [4], a political interview between Margaret Thatcher and Jim Callaghan has been analysed. As a result, the author introduced a classification of turn-taking phenomena where each category is caracterised by the answer to these three questions:

1. Is the attempted speaker switch successful?
2. Is there simultaneous speech?
3. Is the first speaker's utterance complete?

Optimising turn-taking means taking the floor at the right time. Humans are very good at detecting the cues for these timings. In artificial dialogue systems, different kinds of features can be used in order to detect these timings [13]: prosodic features, lexical features, semantic features...etc... and a lot of work has already been done in this direction [25]

2.2 Spoken dialogue systems and incremental behaviour

Spoken dialogue system

A spoken dialogue system (SDS) is an automated application that interacts directly with a human being in natural language. Virtual assistant like Siri (Apple) or Cortana (Microsoft) are good examples of SDSs. They are task-oriented as their goal is to help to user to achieve some task. There also exist a few SDSs that are only used for chatting and companionship.

The classic architecture of an SDS is made of five main modules (Figure 2.1):

1. Automatic Speech Recognition (ASR): transforms the user's audio speech signal into text.
2. Natural Language Understanding (NLU): outputs a conceptual representation of the user's utterance in text format.
3. Dialogue Manager (DM): given the concepts extracted from the user's request, a response (in a conceptual format too) is computed.
4. Natural Language Generation (NLG): transforms the concepts computed by the DM into text.
5. Text-To-Speech (TTS): reads the text outputted by the NLG by using a synthetic voice.

Speech recognition technology is an old problem with long history. During the 1950s, a group of researchers from Bell Labs developed the first technology that is able to recognise digits from speech (in fact, speech perception has been under study since the early 1930s). Then, during the second half of the last century, new advances have made it possible to build ASR solutions with larger vocabulary and with no dependence on the user. In the 1960s, Hidden Markov Models (HMMs) were proved to be useful for speech recognition and they were the most popular technique two decades later. Comercial products using ASR technology had to wait until the 1990s to be finally released in the marked as they reached and interesting vocabulary (even though their accuracy and their delay were far behind the technology we have today). Performances kept improving incrementally until 2009, when Deep Learning algorithms were tested; the Word Error Rate (WER) decreased by 30%. During the last six years, research continued in that direction giving birth to accurate and reactive speech recognition solutions (Google, Nuance, Sphinx, Kaldi...). Therefore, ASR is no longer a bottleneck in the development of spoken dialogue systems, and as we will see later, the delays they offer make it possible to design reactive incremental dialogue systems. Commercial off-the-shelf ASR solutions like Google ASR or Nuance products are able to recognise almost every word in many languages, including named entities. Finally, the ASR output is not only the text that the recognition algorithm figures out to be the best match for the input audio signal, but a list of the N most likely hypotheses and the corresonding confidence scores: it is called the N-Best. For instance, an 5-Best could be:

- 0.965: I would like to book a flight from New-York to Los Angeles

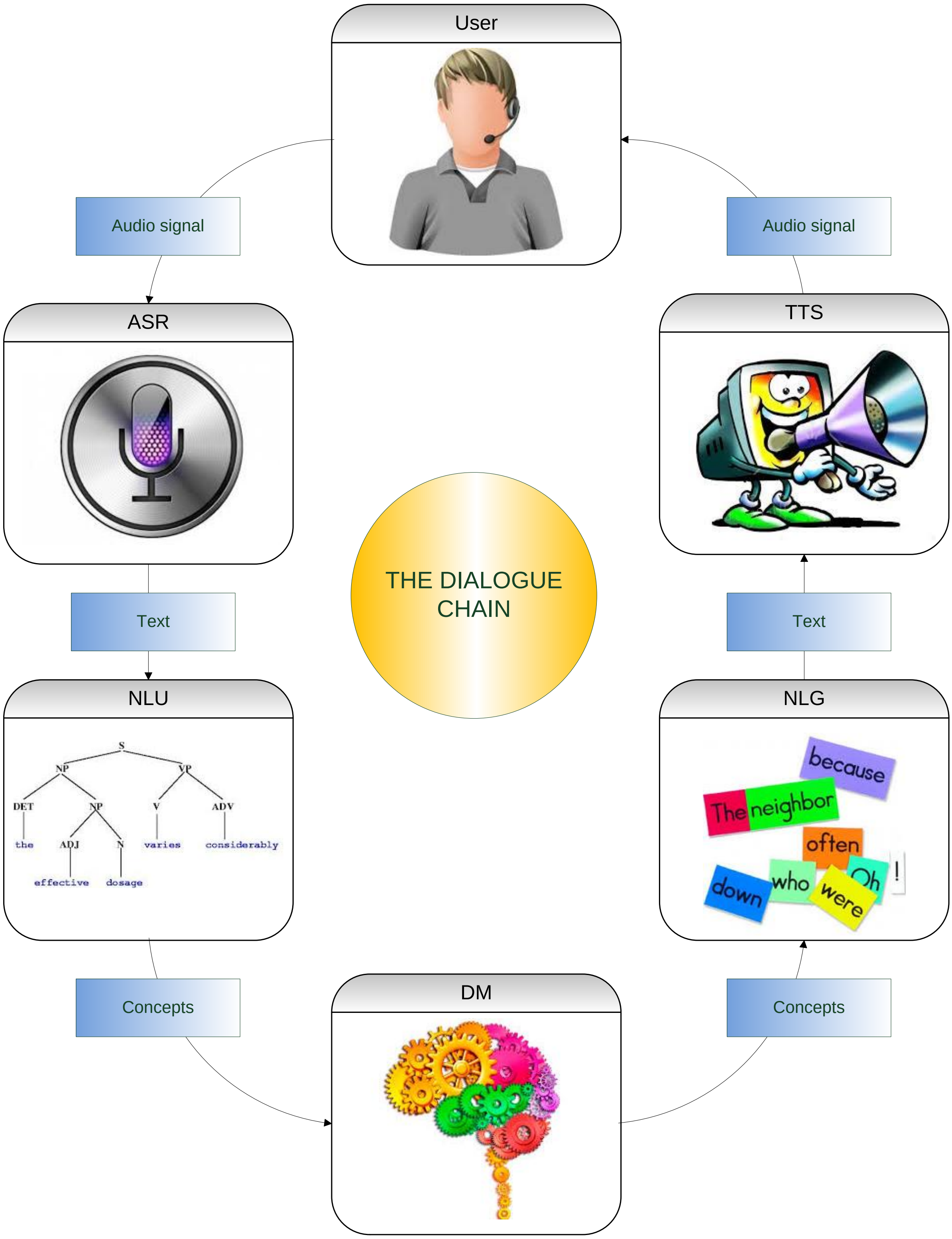


Fig. 2.1 The dialogue chain

- 0.931: I could like to book a flight for New-York to Los Angeles
- 0.722: Cold like a book fly New-York to Los Angeles
- 0.570: Cold like look flight pork to Los Angeles
- 0.441: Cold bike look fly New-York to Los Angeles

The scope of NLU technology is a subfield of Natural Language Processing (NLP) whose scope much wider than the spoken dialogue field. Since the 1950s, researchers have been trying to develop several models and ontologies in order to automatically process natural language with several applications in sight: topic recognition, sentiment analysis, news filtering and analysis, natural speech processing...etc...The ambition manifested during 1950s and the early 1960s quickly had to face reality as the expected objectives were far from being reached. As a consequence, research in this area was significantly slowed down between the 1960s and the 1980s. During the last decade, NLP reasearch has found a second wind thanks to new Machine Learning, bringing them at the heart of lucrative businesses like recommendation and digital marketing. NLU refers to the set of techniques in order to make the machine understand the underlying structure of a text in natural language. To do so, a lexicon as well as an ontology (concepts and the links between them in a specific domain) should be built. Existing dialogue systems are able to interact with the user in the domain they are built for only, however, during the last few years, researchers have been pushing the boundaries of open-domain systems [11]. Therefore, earlier NLU solutions are based on a set of handcrafted parsing rules, however, new statistical-based models have been proven to be more robust and easy to generalise over domains.

Dialogue Management is at the heart of Spoken Dialogue System's current research. A couple of decades ago, for the first time, dialogue has been modeled as Markov Decision Processes (MDPs) problem, hence being solved using reinforcement learning [9]. The dialogue state is generally defined by the whole dialogue history and the set of actions in each state are the dialogue acts that the system can make. In 2007, in order to represent the uncertainty over the user's intent (due to ASR or NLU imperfections), dialogues have been cast as Partially Observable Markov Decision Processes (POMDPs) [42]. This gave raise to the notion of *believe tracking* which objective is to keep a distribution over the possible user intents. In order to encourage research in that direction, a Dialogue State Tracking Challenge (DSTC) has been launched a few years ago [41], bringing new contributions each year. This approach has been shown to provide intersting results, however, the size and the dimensionality of the state space makes it untractable most of the time.

The NLG task is the inverse of the NLU one. It started being used in the 1990s for purposes like financial news summary. A few startups and big companies also provide automatic text generation solutions that are used to quickly produce reports or official letters. The main challenge for such systems is to be able to generate a variety of different words, expressions and sentencs in order for them not to be repetitive and for the result to be as realistic as possible. This is even more crucial when it comes to dialogue systems as they are supposed to simulate a real conversation with the user which is a highly variable task).

During the 1930s, Bell Labs were not only interested in ASR but they also developed a new approach for the inverse task: the TTS (also known as *speech synthesis*). The human speech is broken into small acoustic components that are sequentially pronounced by the system. They built the first machine demonstrating this mechanism: the Voder. As far as this task is concerned, the challenge for the system si to sound as human-like as possible, in terms of phoneme transitions, speech rate and prosody. Even though substantial advances have been accomplished since the Voder, it is still easy to distinguish between a synthetised and a real human voice.

Incremental dialogue systems

The idea of incremental systems goes back to incremental compilers [23]. An incremental compile processes each new instruction independently from the previous ones. Therefore, a local modification of the code does not affect the whole result of the compilation. The idea of processing natural language in an incremental way is first introduced in [43]. Instead of feeding modules with complete utterances, the input is pushed chunk by chunk (500ms of audio signal, one word of a sentence...etc...) making the output change several times before the end of the user's utterance.

Currently deployed dialogue systems have a simple and rigid way of managing turn-taking. The interaction mode they offer is similar to a walkie-talkie conversation as the system waits for the user to finish her utterance before taking the floor and vice-versa (even though some systems allow the user to interrupt the system). Such systems will be referred to as *traditional dialogue systems* in this thesis.

In human to human conversation, the listener does not wait for the speaker to finish his sentence before processing it; it processes it as it is spoken [40]. As a consequence, human beings perform a large panel of turn-taking behaviours while speaking, like backchanneling (*aha*, *yeah*...) or barging-in.

To replicate this behaviour, a new generation of SDSs has been the focus of current research for a few years. An SDS is said to be incremental when it is able to process the user's speech on the fly. The input signal is divided into small chunks and the growing sentence is reprocessed at each new chunk [33].

Advantages of incremental processing

New challenges raised by incremental dialogue processing

The first problem to consider when talking about incremental spoken dialogue systems is the question of ASR *latency*, which is the time needed by the recognition algorithm to provide the text output corresponding to an audio signal. As we said earlier, the ASR accuracy has been a bottleneck in the development of spoken dialogue systems for many years but thanks to recent advances in this field, it is no longer the case. Similarly, incremental dialogue systems require quick responses from the ASR and the delays needed but speech recognition modules have been too long for many years which was a limiting factor in the development of incrmental dialogue systems. However, in the last few years, ASR technology has became reactive enough [27]. Nevertheless, it is important to be aware that there is a tradeoff between the accuracy and the vocabulary size on one hand and the latency on the other hand. Kaldi, which is an ASR solution designed by researchers (and which is mostly used by them), makes it possible to design one's own acoustic and language model as well to set one's own parameters in order to control this tradeoff. On the other hand, more off-the-shelf solutions like Google ASR do not give the user such possibilities (however, accuracy and delays are well balanced for most applications).

If we observe the successive partial results of an incremental ASR module during a user's utterance, it is likely that the progression they follow is not monotonous. In other words, a partial result is not guaranteed to be a prefix of results to come. The following example showing successive ASR results illustrate this phenomenon:

1. Euh
2. I
3. Euh good
4. iPod
5. I would
6. I good bike
7. I would like

This phenomenon is called ASR *instability* (or stability depending on the sources) [36]. This factor is also related to the tradeoff between latency and accuracy as prefering fast ASR over accurate ones can lead to very instable results (the system is not given enough time to check that its results are accurate, thus ending up delivering wrong partial results most of the time), and the vice versa.

This leads to one of the main challenges raised by incremental processing: the ability to *revise* the current hypothesis. All the modules in the dialogue chain are impacted by this problem. As an illustration, suppose that the user interacts with an incremental personal assistant on her phone and makes the following request *Please call the number 01 45 80 12 35*. The last digit is first understood as being 30 and then 35, therefore, if the system is too reactive, there is a risk that it starts calling the wrong number and maybe start uttering the sentence *Ok, calling 01 45 80 12 30*. Afterwards, the system understands 35 instead of 30 hence needing a correction mechanism in order to stop the TTS, to cancel the call, to perform a new one and to provide a new answer to the user. Nevertheless, even though the system at hand is equiped with such a mechanism, using it very often is not an optimal way of managing incremental input as it causes extra delay as well as non-natural behaviour (stopping the TTS and starting again with another utterance). This introduces a similar tradeoff to the one discussed for the ASR module but from the DM perspective: if decisions are taken too quickly, it is likely that some of them are wrong hence activating the correction mechanism. On the other hand, if the DM is slow to take action, then it lacks reactivity and there is no point for it to be incremental. As a consequence, it is important to determine the right moment to commit to the current partial utterance and to take action based on it [29**?**]. This constitues the main objective of this thesis.

Incremental NLG also raise new problems which are illustrated in [3]. In this paper, a system has to describe a car's trajectory in a virtual world. When the latter approaches an intersection where it has to turn right or left (no road straight ahead), then the system utters something like *The car drives along Main Street and then turns...euh...and then turns right*. In this example, the system is sure that the car is going to turn which makes it possible for it to commit to the first part of the sentence with no risk. However, this is not always the case as a new chunk of information from the user can change the whole system's response. In this thesis, the NLG is not incremental as we consider that the DM's response is computed instantly at each new micro-turn (event though it is not necessarily stabe and it can vary from micro-turn to micro-turn). Finally, in purely vocal applications, computing the NLG results incremental does not make much sense as the user's and the system's utterances do not overlap most of the time [30]. However, this is an interesting behaviour as far as multimodal applications are concerned.

Building an incremental TTS module can also be very tricky. In order for the synthetic voice to be the most human-like as possible, prosody should be computed accurately and to do so, the sentence's structure and punctuation have to be taken into account. This information is no longer given in the case of incremental TTS or it arrives too late.

2.2.1 Existing architectures

Sequential paradigm

A general abstract model of incremental dialogue systems has been introduced in [33]. In this approach, the dialogue chain is maintained and each one of the five components is transformed into its incremental version. We will refer to this view of incremental dialogue systems as the *sequential paradigm*.

Each module is composed of three parts, the Left Buffer (LB), the Internal State (IS) and the Right Buffer(RB). As described in Section 2.2, each module is also characterised by the type of input it processes as well as the type of output it computes. In incremental dialogue, all these data flows have to be divided into small chunks which are called Incremental Units (IU) [33]. For example, the audio signal that is given as an input to the ASR module can be divided into 500ms chunks that are processed one by one. Each IU is first added to the LB, then it is taken by IS for processing and once a result is available, a new IU of a new kind is outputted in the RB. As the RB of one module if the LB of the following one in the dialogue chain, the data propagation through the dialogue system is insured.

It is important to note that a new IU in the LB does not necessarily imply that a new IU will be pushed into the RB on top of the ones that already existed there. An example given in [33] is the following: suppose the user utters the number *forty* which processed incrementally, then first the ASR outputs *four* and then *forty*. As a consequence, the second hypothesis does not complete the first one but it cancels it. This phenomenon will be discussed more in details in Chapter 5.

Adopting this paradigm is a natural way of enhancing traditional dialogue systems with incremental capabilities. It is interesting from a computational and design point of view as the different tasks are separated. Therefore, one is able to evaluate the different components independently [2] and have a global view on which area still needs improvement.

Multi-layer paradigm

The problem of dialogue management in traditional dialogue systems can be formulated as follows: at each dialogue turn, given the dialogue context (including the last user's utterance), what is the right dialogue act to perform? In the incremental frame, this definition no longer holds as dialogue acts are no longer attached to dialogue turns. Therefore, one way to tackle the problem is to split the dialogue management task in two components, the high-level and the low-level handlers. This paradigm is directly motivated by Austin's, Searl's and Clark's contributions discussed in Section 2.1.1 as the high-level module handles illocutionary acts (the communicative track) whereas the low-level one manages locutionary acts (the meta-communicative track).

As reported in [20], this approach is more in alignment with results in the psycholinguistic field. The phenomena observed in the meta-communicative track are complex, and the interaction happen on multiple levels, not always following the classical dialogue chain. Having a separate module for handling these phenomena is therefore a more natural paradigm.

Switching from the traditional dialogue management approach to the incremental one is also a transition from discrete time to continuous time, from a synchronous to an asynchronous processing [28]. The low-level module is continuously (approximated by a high frequency processing in computers) listening to the outside world and waiting for events that might be interesting to communicate to the high-level handler. On the other hand, the latter communicates actions (dialogue acts) and it is the role of to the low-level module to choose whether to retrieve them to the user or not as well as choosing the right moment in case it decides to speak.

Finally, starting from a traditional dialogue system, it is more easier and straightforward to transform it into an incremental one if one adopts this paradigm. Adding an extra low-level module to the dialogue manager is enough [37, 16]. At each new incremental input, this module sends the whole partial utterance from the beginning of the current turn to the dialogue manager and gets its response. Based on that and eventually some other features, it decides whether to take the floor or not. As most of the requests sent to the dialogue manager are "fake" as they are not meant to be acted on, they should not affect the dialogue context. Therefore, whether multiple instances of the dialogue manager are used, whether the dialogue context is saved and restored at each new request, unless the low-level module decides to take the floor.

2.3 Reinforcement Learning

2.3.1 Definition

Reinforcement Learning (RL) is a sub-field of machine learning where an agent is put into an environment to interact with, and figure out through the process of *trial and error*, what the best actions to take are, given a reward function to maximise [39] (see Fig. 2.2).

Formally, the agent is cast as a Markov Decision Process (MDP) which is a quintuple $\mathscr{M} = (\mathscr{S}, \mathscr{A}, T, R, \gamma)$ where:

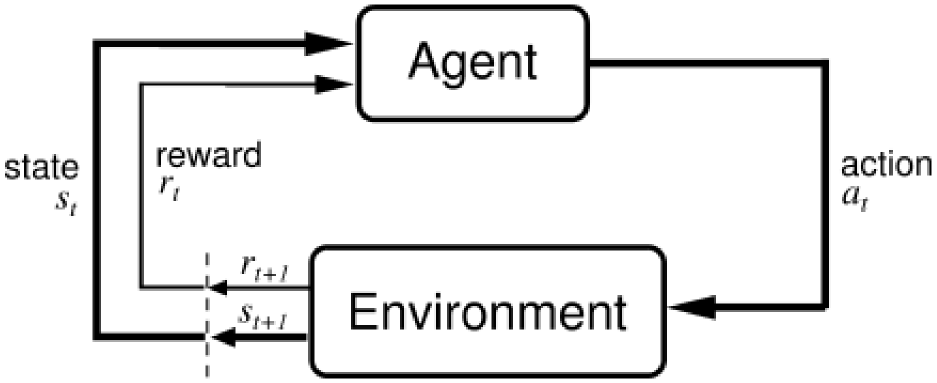


Fig. 2.2 The interaction between the agent and the environment in reinforcement learning

- $\mathscr{S}$ is the *state space*. At each time step t, the agent is in some state $s_t \in \mathscr{S}$.
- $\mathscr{A}$ is the *action space*. At each time step t, the agent decides to take action $a_t \in \mathscr{A}$.
- T is the *transition model*. It is the set of probabilities $\mathbb{P}(s_{t+1} = s' | s_t = s)$ for every $(s, s') \in \mathscr{S}^2$.
- R is the *reward model*. If r_t is the immediate reward due to taking action a_t in state s_t, then R is the set of distributions of r_t for every $(s_t, a_t) \in (\mathscr{S}, \mathscr{A})$.
- $\gamma \in [0, 1[$ is referred to as the *discount factor*. In the RL framework, the aim of the agent is not to maximise the immediate reward but the *expected return*, where the return R_t is defined as follows:

  $$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

  Therefore, when $\gamma = 0$, the agent maximises the immediate reward only and when γ tends towards 1, the agent maximises the sum of all the future rewards. More generally, the parameter γ controls how far-sighted is the agent in terms of future rewards.

A *policy* $\pi : \mathscr{S} \rightarrow \mathscr{A}$ is a mapping between the state space and the action space. An agent is said to follow the policy π when for each time t, it takes the action $a_t = \pi(s_t)$. A policy can also be stochastic, in which case, $\pi(s, a)$ denotes the probability of choosing action a when the agent is at state s. A key aspect of MDPs is the *Markov property*. Being at state s is the only information necessary to predict the future expected return, and no information about the past is necessary. Therefore, given a policy, each state $s \in \mathscr{S}$ is given a value $V^{\pi}(s)$ which is the expected return for being at this state and following the policy π afterwards:

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

Another interesting quantity is the expected return knowing the current state but also the next action, after which π is followed. This is referred to as the Q-function:

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

Given the definition of R_t, we can notice that

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}[R_t | s_t = s, \pi] \\
&= \mathbb{E}[r_t + \sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k+1} | s_t = s, \pi] \\
&= \mathbb{E}[r_t + \gamma R_{t+1} | s_t = s, \pi] \\
&= \mathbb{E}[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s, \pi]
\end{aligned}$$

This is known as the Bellman equation and it can also be written for the Q-function as follows

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a, \pi]$$

A natural question that can be asked at this point is: how do we compute these values? In reinforcement learning, this is known as the *evaluation problem*. The transition model T and the reward model R are the elements that define the dynamic of the MDP. If they are known, V^{π} can be directly computed. If we call $P^a_{ss'} = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$ and $R^a_{ss'}$ the reward of chosing action a on state s and landing on s', we can write:

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}[R_t | s_t = s, \pi] \\
&= \sum_{a \in \mathscr{A}} \pi(s,a) \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \\
&= \sum_{a \in \mathscr{A}} \pi(s,a) \mathbb{E}[r_t + \gamma R_{t+1} | s_t = s, a_t = a, \pi] \\
&= \sum_{a \in \mathscr{A}} \pi(s,a) \sum_{s' \in \mathscr{S}} P^a_{ss'} (R^a_{ss'} + \gamma \mathbb{E}[R_{t+1} | s_{t+1} = s', \pi]) \\
&= \sum_{a \in \mathscr{A}} \pi(s,a) \sum_{s' \in \mathscr{S}} P^a_{ss'} (R^a_{ss'} + \gamma V^{\pi}(s'))
\end{aligned}$$

It is possible to define an order over the policies. Saying that π_1 is better that π_2 means that for all the states s, $V^{\pi_1}(s) \geq V^{\pi_2}(s)$. It can be shown that there exists at least one policy that is better that all the others: it is called the *optimal policy* (π^*). To simplify the notations, V^{π^*} will be referred to as V^* and it defined as

$$\forall s \in \mathscr{S},\ V^*(s) = \max_{\pi} V^{\pi}(s)$$

Similarly, we can define Q^* as

$$\forall (s,a) \in \mathscr{S}x\mathscr{A},\ Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

The aim of reinforcement learning is to learn the optimal policy. Similarly to what has been shown for V^{π}, if the transition and the reward models are known, the Bellman equation corresponding to V^* can be written with respect to these models:

$$V^*(s) = \max_{a} \sum_{s' \in \mathscr{S}} P^{a}_{ss'}(R^{a}_{ss'} + \gamma V^*(s'))$$

A similar form can be also be shown about the Q-function

$$Q^*(s,a) = \sum_{s' \in \mathscr{S}} P^{a}_{ss'}(R^{a}_{ss'} + \gamma \max_{a' \in \mathscr{A}} Q^*(s',a'))$$

A set of *Dynamic Programming* methods exist in order to efficiently solve these kinds of equations and come up with the optimal policy given the transition and the reward model (knowing Q^* implies knowing π^* as the latter is the greedy policy with respect to the former Q-function). However, even though this kind of approaches are theoretically interesting, they only have a few practical applications as most of the times, T and R are unknown.

Fortunately, there are other methods that have been shown to be successful in learning the optimal policy without any prior knowledge about the MDP dynamics as they completely learn by trial and error. The first set of approaches are called *Monte Carlo*; the Q-function is directly evaluated by calculating the mean of the returns R_t for each coupe (s,a). Nevertheless, this method requires the current learning episode to be finished before updating the Q-function. Moreover, acting as such means evaluating V^{π} (or Q^{π} according to the chosen approach) with π being the policy currently followed by the agent. So, how do we compute the optimal policy? This raises the problem of *control* wich goes along with the question of evaluation introduced earlier. In fact, as we keep changing the Q-function values because of new agent experience, we also keep upating the policy to make it greedy with respect to the new evaluation results. Fig. 2.3 illustrates this idea.

The second and the most widely used approach is called *Temporal Difference Learning* (TD-Learning). The algorithms that belong to this category do not wait for the end of the episode to update the Q-function like in the case of Monte-Carlo, instead, they use other

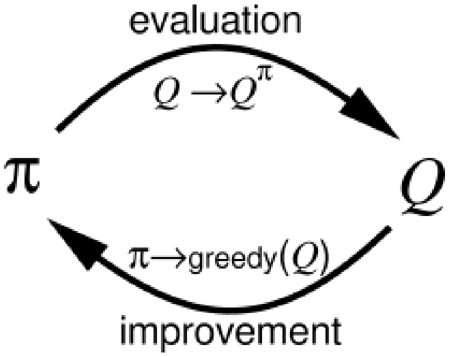


Fig. 2.3 The alternation between evaluation and control

estimated values for other states and other actions; this is known as *bootstraping*. The most famous TD algorithms are *sarsa* and Q-Learning. The first one updates $Q(s_t, a_t)$ at time t by choosing the next action given a policy derived from Q (ε-greedy for example), obtaining an immediate reward r_t, moving to state s_{t+1} and choosing a_{t+1} using a policiy derived from Q again, then updating Q as follows (α being an update parameter):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

The idea behind this way of boostraping is that the Q is updated at each step by considering that it will follow a policy derived from Q during the next steps; this is an *on-policy* algorithm. On the contrary, Q-Learning [**?**] is an *off-policy* algorithm as it directly estimates Q^* by performing the following update at each step:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

2.3.2 Application to dialogue systems

Reinforcement learning has been first applied to dialogue systems in [22] and since then, it has the leading machine learning framework in the field. The dialogue state at time t is generally determined by the history of dialogue acts since the beginning of the dialogue. At each turn, the set of actions is made of all the possible answer at that time. Partially Observable Markov Decision Processes (POMDPs) are also widely used. In this framework, the dialogue state is replaced by a distribution over all possible states which is a more natural way of modeling uncertainty, however, they are more complex and more difficult to scale [21].

In the field of incremental dialogue and turn-taking management, supervised learning is more common. The main problem tackled by researchers is the identification of the exact moments where the system should take the floor in order to achieve smooth turn-taking [29, 25]. Binary classifiers are used and the features they are fed are of different natures: lexical, semantic, prosodic...etc...However, a few papers tackled this problem by using reinforcement learning.

[15] used reinforcement learning while considering prosodic features only. Backchanneling for example can be performed by humans independently from the meaning. The cost function (negative reward) is taken as gaps and overlaps, hence following Sack's principle discussed in Section 2.1.2.

[8] adopted a complementary approach where only the semantic content of the user's utterance is taken into account (hierarchical reinforcement learning is used). In human conversation, it is more likely for the listener to react right after a relevant information. Similarly, in the case of a restaurant finding spoken dialogue system, the system should react right after understanding the restaurant's type or price range. In this work, the information pertinence is measured by the Information Density (ID). Therefore, the more the ID is high during system actions, the more reward it gets.

Instead of trying to minimise gaps and overlaps, the reward function can be designed in a way to optimise dialogue duration and task completion like it is the case in [35]. The system in this paper learns optimal initial turn-taking, in the sense that when a silence is detected, the dialogue participant that has the most relevant thing to say takes the floor first. Like in the previous paper, only semantic features are considered.

A third approach to optimise turn-taking in spoken dialogue systems is to directly try to imitate human behaviours. In [17] Inverse Reinforcement Learning is used to infer a reward function directly from user trajectories in a gathered dialogue corpus. Therefore, the reward function automatically incorporates objective and subjective dialogue quality criteria. The authors have made the choice not to consider lexical and semantic features, but rather to limit their work to timing and prosody signals.

2.3.3 Dialogue simulation

A couple of decades ago, as dialogue systems started to become a popular research fields, the need for evaluation means in order to assess their quality started getting more and more important. Therefore, researchers turn to user simulation methods (also referred to as user modeling). In [9], some of the advantages of these techniques are depicted: reduced cost with automatic evaluation of a large number of dialogues, less error risk, easy modeling of different user populations, possibility of using the same user model across different concurrent dialogue systems for comparison and providing a tool to quickly generate corpora for machine learning techniques at a low cost. Nevertheless, the authors recognise that user simulation cannot totally replace interactions with real users in the process of designing reliable dialogue systems: *However, we believe that tests with human users are still vital for verifying the simulation models..*

Simulating users accurately is a challenging task as their behaviours vary considerably from a person to another and moreover, the same user can change her preferences over time (concept-drift) [32]. Evaluating a user simulator and whether it handles such variability or not is a research track in itself [26] and the qualities required are of different kinds. The trained user simulator should be consistent with the data that has been used for the training and the sequence of dialogue acts generated should be coherent. In addition, when it is used in turn to train a data-driven dialogue strategy, the quality of the latter is also an evaluation criteria. Also, it is important that the results obtained in simulation give strong indications about the behaviours with real users while being task independent and automatically able to automatically compute assessments.

User simulation is useful during the conception phase of a dialogue system. However, training the simulator from data needs the dialogue system to be conceived already. Therefore, trying to come up with a simple model with only a few parameters is not always a bad idea as it has been proven to achieve good results as well [31].

User simulator is also quite similar to the dialogue management task. As a consequence, it is legitimate to ask the following question: why not use reinforcement learning to train user simulators? The answer is that in the case of dialogue management, it is easier to come up with a reasonable reward function: task completion, dialogue duration, subjective evaluation...etc... When it comes to user simulation, the objective function is how well a real user is imitated which is impossible to handcraft. Fortunately, there exists a framework where the reward function is automatically inferred from data which is particularly useful here: inverse reinforcement learning [6].

When it comes to incremental dialogue systems, the only existing user simulator in our knowledge is the one described in [38]. Its state is update every 10 ms. However, the *ASR instability* phenomenon is not replicated, that is to say that the ASR hypothesis construction is monotonous whereas in reality, it is not the case. When a new audio signal increment is heard by the ASR, the output can be partially or totally modified. In this simulator, only the simple case where a new increment is added to the output is modeled.

2.4 Issues and motivation

A study led by the Market Intelligence and Consulting Institute shows that the market of artificial intelligence should grow exponentially in the next decade. Virtual assistants play the main role in this domain and they are predicted to multiply their market share by 2.5. Moreover, the market is expected to multiply its turn over by 14 (30% growth per year on average). This shows that SDSs have reached a level of maturity that enables them to significantly interest the market. Therefore, designing robust, reliable and user-friendly spoken dialogue systems raises an important issue.

Nevertheless, even though virtual assistants like Siri or other more domain specific spoken dialogue systems can offer a quite reasonable quality of service for certain tasks, the quality of the interaction is still poor, making room for improvement in different areas:

- The ASR module is not perfect and the errors that it engenders are not well handled by the DM.
- The vocabulary is very restrained hence users often use off-domain words, especially those who are not familiar with the system.
- Current Natural Language Processing (NLP) techniques still do not cover a lot of discourse formulations, therefore NLU modules are often too simple to handle all the situations encountered in real dialogue.
- Current SDSs do not adapt to the user's profile neither to groups of users's particular behaviours (accent, culture, specific expressions...).
- Turn-taking is handled in a walkie-talkie manner which is too simple compared to the reality of dialogue.

In this thesis we focus on the last point: turn-taking capabilities improvement. We identified two research streams:

1. Barge-in points identification to achieve smoother and more human like turn-taking. When a barge-in point is detected, it can be interpreted as an end-point so that the system can take the floor in a more reactive way, or as a suitable point for backchanneling. In these kind of studies, prosodic features are crucial but they can also be mixed with lexical and semantic ones.
2. Turn-taking optimisation to improve dialogue efficiency (generally measured through dialogue duration and task completion). The main objective is to improve error handling by reporting errors quickly, and to improve reactivity in general as the system can respond as soon as it has enough information to do so. Unlike the previous stream, semantic features have more importance here.

This thesis is a contribution to the second research stream. First we try to understand what is turn-taking in human-human interaction, then we ask ourselves what phenomena can be replicated in human-machine dialogue in order to offer a better error-handling and an increased efficiency in general.

Chapter 3

Turn-taking taxonomy

3.1 Taxonomy presentation

Turn-taking in dialogue refers to the act of taking the floor by one participant, here called the Taker (T). Two cases can be distinguished; either the other participant, here called the Giver (G), is already speaking or not (the denomination Giver is more adapted to the case where it has the floor, but we keep it as a convention for the other case). In the first case, turn-taking either gives birth to a barge-in where G stop speaking or to a backchannel, feedback or comment and it that case, G keeps talking. If G does not have the floor, T is in a situation of initial turn-taking.

The taxonomy we introduce here is based on two dimensions: *the quantity of information that G has already injected in the dialogue* and *the quantity of information that T tries to inject by taking the floor.* The different levels of information for each dimension are described on Table 3.1.

Table 3.1 *Taxonomy labels*

G_NONE	No information given
G_FAIL	Failed trial
G_INCOHERENT	Incoherent information
G_INCOMPLETE	Incomplete information
G_SUFFICIENT	Sufficient information
G_COMPLETE	Complete utterance
T_REF_IMPL	Implicit ref. to G's utterance
T_REF_RAW	Raw ref. to G's utterance
T_REF_INTERP	Reference with interpretation
T_MOVE	Dialogue move (with improvement)

Table 3.2 describes the taxonomy where turn-taking phenomena (TTP) are depicted. The rows correspond to the levels of information added by G and the columns to the information

Table 3.2 *Turn-taking phenomena taxonomy. The rows/columns correspond to the levels of information added by the floor giver/taker.*

	T_REF_IMPL	**T_REF_RAW**	**T_REF_INTERP**	**T_MOVE**
G_NONE	FLOOR_TAKING_IMPL			INIT_DIALOGUE
G_FAIL	FAIL_IMPL	FAIL_RAW	FAIL_INTERP	
G_INCOHERENCE	INCOHERENCE_IMPL	INCOHERENCE_RAW	INCOHERENCE_INTERP	
G_INCOMPLETE	BACKCHANNEL	FEEDBACK_RAW	FEEDBACK_INTERP	
G_SUFFICIENT	REF_IMPL	REF_RAW	REF_INTERP	BARGE_IN_RESP
G_COMPLETE	REKINDLE			END_POINT

that T tries to add. In order to describe each one of them in detail, we will proceed row after row.

G_NONE G does not have the floor, therefore, T takes the floor for the first time in the dialogue. This can be done implicitly by performing some gesture to catch G's attention or by clearing her throat for instance (FLOOR_TAKING_IMPL). On the other hand, she can start speaking normally (FLOOR_TAKING_EXPL).

G_FAIL G takes the floor for long enough to deliver a message (or at least a chunk of information) but T does not understand anything. This can be due to noise or to the fact that the words and expressions are unknown by the T (other language, unknown cultural reference, unknown vocabulary...). T can interrupt G before the end of his utterance as she estimates that letting him finish it is useless. This can be done implicitly (FAIL_IMPL) using a facial expression (frowning), a gesture or uttering a sound:

G: Cada hora that I spend here is ...
T: ...what?

It can also by uttering explicitly that G's utterance is not clear so far (FAIL_RAW):

G: <noise> has been <noise> from...
T: ...sorry, I can't hear you very well! What did you say?

Finally, T can interrupt G by trying to provide a justification to the fact that G needs to repeat, reformulate or add complementary information in his sentence (FAIL_INTERP). For example:

G: Freddy was at the concert and ...
T: ...who is Freddy?

G_INCOHERENCE T understands G's message and detects and incoherence in it, or between that message and the dialogue context. G can make a mistake like *I went swimming from 10 am until 9 am*, *First, go to Los Angeles, then go south to San Francisco...* or be unaware of the dialogue context: *You should take line A...* while line A is closed that day. Again, this can be done implicitly (INCOHERENCE_IMPL) by adopting the same behaviours as in the case of G_FAIL, or explicitly (INCOHERENCE_RAW).

G: Investing in risk-free instruments like stocks is one of the ...
T: ...that is nonsense.

T can also explain the reasons she thinks this is not coherent (INCOHERENCE_INTERP):

G: I will visit you on Sunday and then ...
T: ...but you are supposed to be traveling by then!

G_INCOMPLETE G's utterance is still incomplete (and G is still holding the floor) but all the information given so far is coherent. T can perform a backchannel by nodding her head for example or by saying *Aha* or *Ok* for example (BACKCHANNEL). This gives G a signal that he is being understood and followed, thus encouraging him to keep on speaking. T can also choose to repeat a part of G's sentence for confirmation (FEEDBACK_RAW). If this part is correct, G continues to speak normally (or sometimes explicitly confirms by adding a *yes* to his sentence):

G: My number is 01 45...
T: ...01 45
G: 12 25
T: 12 29
G: no, 12 25
T: ok, 12 25

Another kind of feedback is by adding some related information to G's incomplete utterance (FEEDBACK_INTERP), for example:

G: I went to see the football game yesterday...
T: ...yeah, disappointing
G: ...with a friend, but we did not stay until the end.

G_SUFFICIENT G has not finished talking, yet, all the information that T needs to answer has been conveyed. If G is listing a few options, T can perform a gesture meaning that she is interested in the last option uttered (REF_IMPL). She can also do it explicitly (REF_RAW):

G: You can book for an appointment, on Monday afternoon, Tuesday morning, Wednesday afternoon...
T: Oh Yeah! That would be great.

T can also add comments related to her choice, once selecting an option (REF_INTERP):

G: We have apple juice, tomato juice...
T: Oh Yeah! Tomato juice is my favorite, plus, my doctor advised to have it.

In the case of goal-oriented dialogue, G keeps talking even though he conveyed all the necessary information for T to formulate an answer. T can choose to interrupt him (BARGE_IN_RESP) though making the dialogue shorter (this can be viewed as a rude move in some cases):

G: I want to book a six person table tomorrow at 6 please, I was wondering if it is possible as ...
T: Sure, no problem. Can I have your phone number please?

G_COMPLETE G has finished his utterance. If T thinks that some more information needs to be provided, she can perform a gesture or adopt a facial expression to communicate that (REKINDLE), making G take the floor again and provide further information. This can also be done explicitly and it will be considered as a new dialogue turn, as well as T providing new information to make the dialogue progress.

G: How many friends of yours are coming with us tomorrow?
T: Two, hopefully.

3.2 Discussion

This taxonomy is aimed to clarify the notion of turn-taking. In human-human conversation, this translate into a rich set of behaviour that we try to depict and classify given two criteria. Compared to existing classifications of turn-taking behaviours, an important part is given to the semantic content of G's and T's utterances (and other cues like gestures and facial expressions) as well as the reasons that pushed T to take the floor given this information.

A big part of research in incremental dialogue systems and turn-taking optimisation has mainly focused on endpoint detection [29] and smooth turn-taking. Therefore, their objective is to replicate the phenomenon labeled here as BARGE_IN_RESP. Some other studies focus on backchanneling and feedback, often neglecting the semantic part of the dialogue participants utterances and focusing exclusively on prosody and acoustic features.

Chapter 4

Architecture

4.1 Description

4.1.1 Overview

The dialogue chain is made of five modules: ASR, NLU, DM, NLG and TTS (Chap. 2). In the architecture introduced here, they are split in two groups: those forming the *client* and those constituting the *service*. The ASR and the TTS are necessarily included in the client and the DM in the service. The NLU and the NLG can fit in both categories. This terminology has been borrowed to the computer network field where the client can refer to the user and to the application that interacts directly with the user in order to gather useful data for the interaction at the same time. Similarly, the server refers to the application that is in charge of handling user's requests, as well as the remote machine it is deployed on. In the case of dialogue systems, both parts can be embedded in the same device and they can also be distributed in two machines.

Viewing traditional dialogue systems from this point of view translates into a ping-pong game, where the client sends a request which is processed by the service, and the latter sends a response back. The question we ask ourselves here is how to break this rigid mechanism in order to make the system able to process the user's speech incrementally. This chapter shows how, by starting from this new view of dialogue systems instead of the sequential one (dialogue chain), an incremental dialogue system can be derived from a traditional one at minimal cost. Moreover, in the resulting architecture, the turn-taking decision center is separated from the DM.

As illustrated in Fig. 4.1, a new interface is inserted between the client and the service [16]. We call this new module the *Scheduler* (this denomination is borrowed from [18]). It can be deployed in the same machine as the client, as the service or in a dedicated server. The

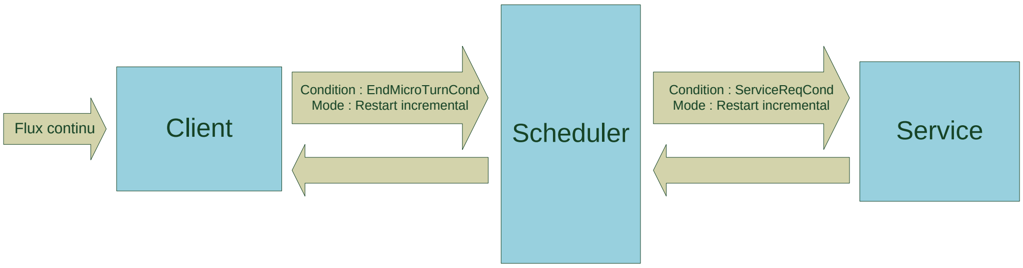

Flux continu
Client
Condition : EndMicroTurnCond
Mode : Restart incremental
Scheduler
Condition : ServiceReqCond
Mode : Restart incremental
Service

Fig. 4.1 The Scheduler: an interface between the client and the service

objective to make the set {Scheduler+service} behave like an incremental dialogue system from the clients point of view, without modifying the initial functioning of the service. By doing so, we provide a framework that can transform any dialogue system in its incremental version just by adding a new layer.

This alternative way of designing incremental dialogue systems also has the advantage of clearly separating turn-taking management from the rest. As it will be seen along this thesis, turn-taking strategies will be implemented exclusively in the Scheduler (the rest of the system remaining the same). Our ultimate goal is to make this module learn optimal turn-taking behaviours by itself.

4.1.2 Time sharing

There are four types of human machine interfaces, given whether both communication channels (from the user to the system and the other way around) are punctual or discrete:

- **Discrete User/Discrete System:** This is the most basic way of human computer interaction. For example, the user hits a button and she immediately gets a response back. This is still widely used in many applications like browsers (when basic surfing, without watching videos or listening to audio).

- **Discrete User/Continuous System:** Most of the currently deployed vocal platforms operate in this mode as they asked the user for DTMF inputs while they use natural language to provide instructions (using pre-recorded audio or speech synthesis).

- **Continuous User/Discrete System:** The application Shazam would be a good example of this communication mode. The user starts playing a song and the application listens. Once the latter recongnises it, it instantly displays the title and the singer on the screen.

- **Continuous User/Continuous System:** Spoken dialogue systems operate in a continuous/continuous mode. The communication signal from both sides is continuous.

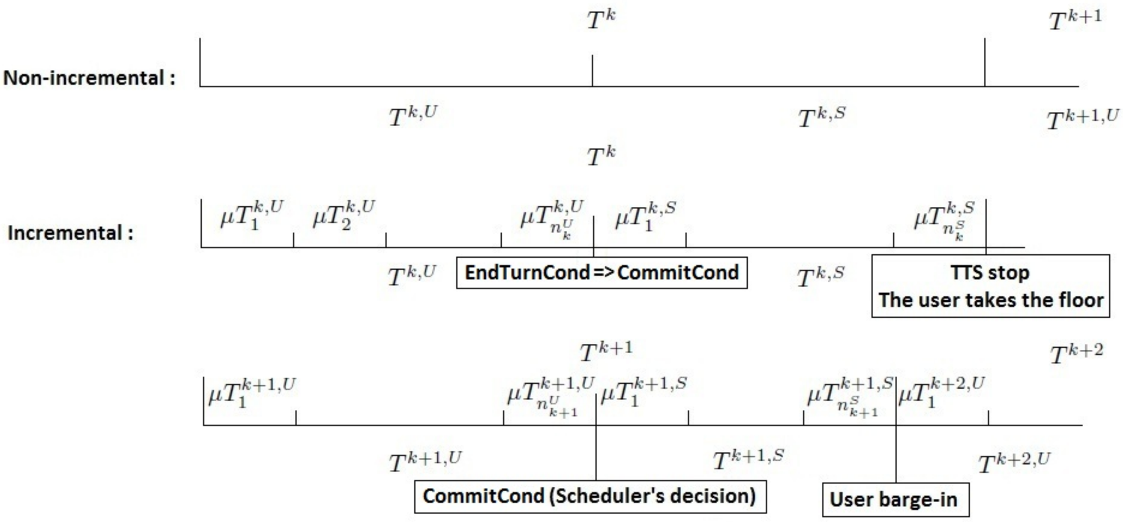


Fig. 4.2 Time sharing in traditional and incremental settings

For a system to be incremental, the user has to be continuous, therefore, the first and second communication modes are out of the scope of this thesis.

In traditional dialogue systems, time is shared in a ordered and clear manner. The dialogue is a simple sequence of turns T^1, T^2..., a turn being the time interval in which a user's utterance followed by the system's response takes place, or the opposite (depending whether the system adopts a user initiative or a system initiative strategy at each time). For illustration and to simplify the notation, we will suppose that our system belongs to the first category, therefore, each turn is divided into two smaller time intervals, the user turn $T^{k,U}$ and the system turn $T^{k,S}$: $T^k = T^{k,U} \cup T^{k,S}$ (Fig. 4.2).

In this chapter, a few conditions are defined to precisely describe time allocation between the system and the user. The *activation time* of a condition refers to the exact moment when it goes from false to true. *EndTurnCond* is the condition that ends a user turn, it is generally assimilated to a long silence [29, 44].

In incremental settings, this time sharing formalism does not hold anymore and a new condition should be defined: *EndMicroTurnCond* (with *EndTurnCond* ⇒ *EndMicroTurnCond*). The time interval separating two activation times of *EndMicroTurnCond* is called a *micro-turn*. As a consequence, the turn $T^{k,U}$ can be divided into $n^{k,U}$ micro-turns $\mu T_i^{k,U}$: $T^{k,U} = \bigcup_{i=1}^{n^{k,U}} \mu T_i^{k,U}$. The pth *sub-turn* of turn $T^{k,U}$ is defined as $T_p^{k,U} = \bigcup_{i=1}^{p} \mu T_i^{k,U}$.

The request that user makes during $T^{k,U}$ is referred to as Req^k and the corresponding response is $Resp^k$. This architecture does not process incremental units like in [33], instead, at each new micro-turn, it will take the whole information available since the beginning of the turn (at the pth micro-turn, all what the user uttered during $T_p^{k,U}$). This *partial request* is called Req_p^k.

4.1.3 The Scheduler

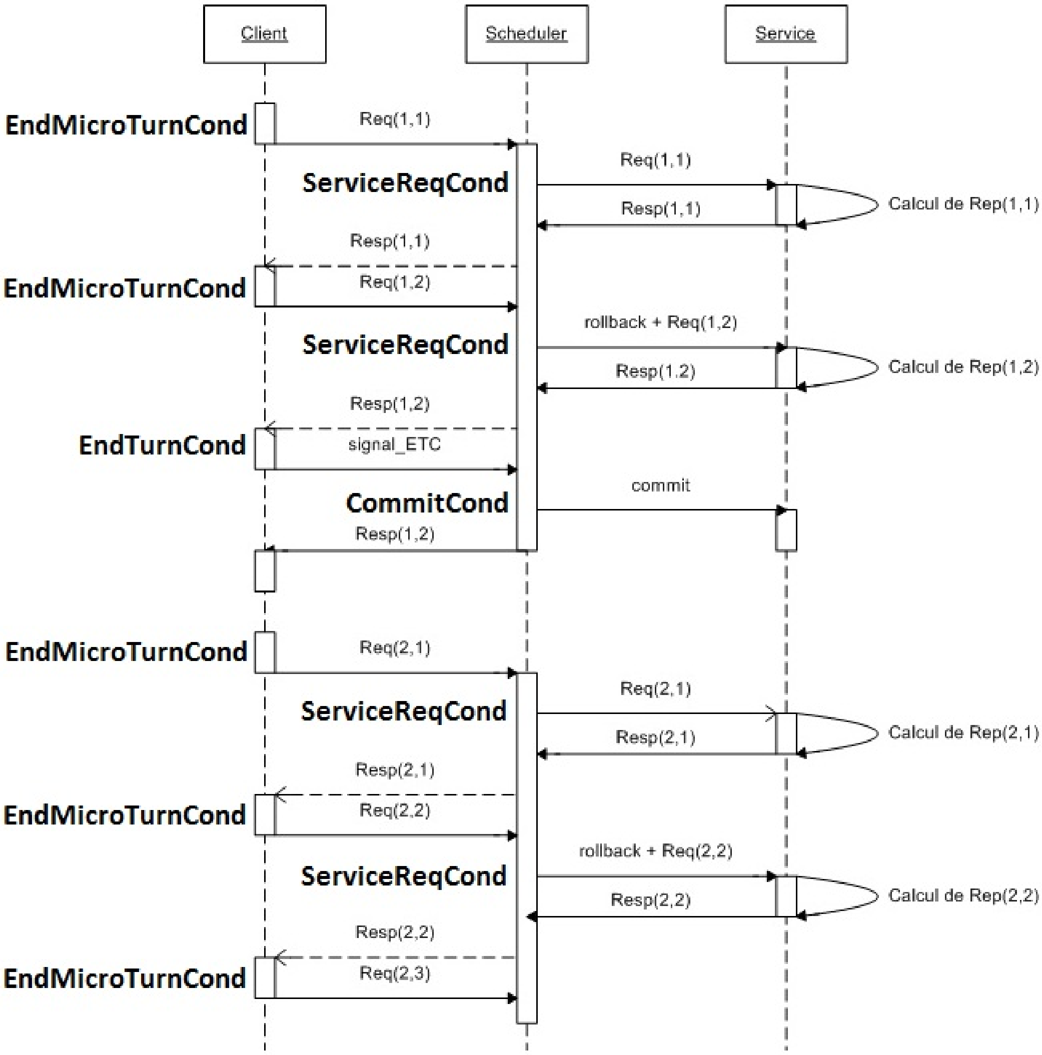
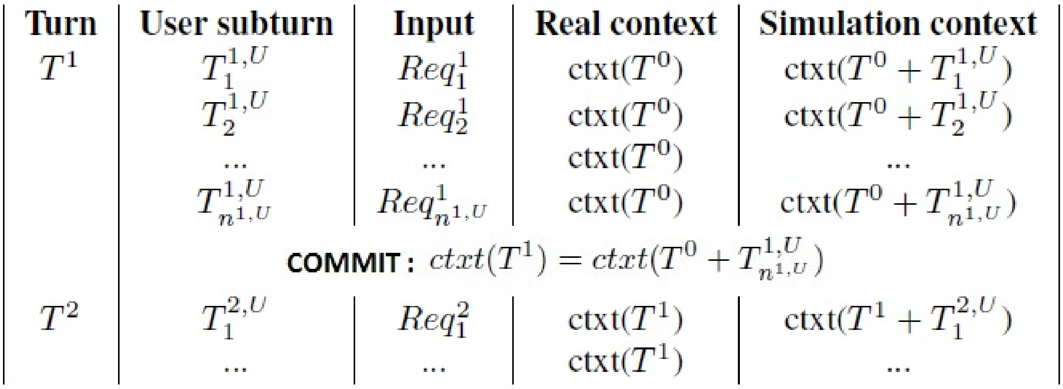


Fig. 4.3 Incremental behaviour with the Scheduler

During the pth micro-turn of the kth user turn, the client sends Req_p^k to the Scheduler. The latter has to decide whether to send it to the service or not and the corresponding condition is called *ServiceReqCond*. An good example is *ServiceReqCond* = ($Req_p^k = Req_{p-1}^k$) as sending the same request twice is useless. Then, the service provides the corresponding response $Resp_p^k$ and the Scheduler stores it. The key idea of this architecture is that the Scheduler decides whether to retrieve this response to the client (making it take the floor through the TTS) or not (waiting for more information to come from the client. This decision can also be forced by the client when sending an end of turn signal *signal_ETC*, like a long enough silence for instance. The most interesting fact about the Scheduler is that it is able to decide when to take the floor without waiting for *signal_ETC*, the corresponding condition is called *CommitCond*. The Scheduler functioning of over time is illustrated in Fig. 4.3.



Turn	User subturn	Input	Real context	Simulation context
T^1	$T_1^{1,U}$	Req_1^1	ctxt(T^0)	ctxt($T^0 + T_1^{1,U}$)
	$T_2^{1,U}$	Req_2^1	ctxt(T^0)	ctxt($T^0 + T_2^{1,U}$)
	...	...	ctxt(T^0)	...
	$T_{n^{1,U}}^{1,U}$	$Req^1_{n^{1,U}}$	ctxt(T^0)	ctxt($T^0 + T_{n^{1,U}}^{1,U}$)
	COMMIT: $ctxt(T^1) = ctxt(T^0 + T_{n^{1,U}}^{1,U})$			
T^2	$T_1^{2,U}$	Req_1^2	ctxt(T^1)	ctxt($T^1 + T_1^{2,U}$)
	...	...	ctxt(T^1)	...

Fig. 4.4 Double context management: real and simulated

Nevertheless, this approach raises on technical problem. Most of the requests that are made to the service are only aimed to see what would be its response for certain partial utterances and they are not used in the dialogue. However, they might modify the dialogue state in the service which is a side effect to be avoided. As a consequence, two dialogue contexts are maintained:

- **The real context:** The dialogue context as traditionally used in dialogue systems. Contains the data and the variables that are aimed to last and be used in the rest of the dialogue.
- **The simulated context:** A rough copy of the real context, at the pth micro-turn, $Resp_p^k$ could be useful for the dialogue or not. Therefore, only this context is modified at the first place, the Scheduler decides later whether to keep the changes in the real context or not.

These dialogue contexts are managed by two actions performed by the Scheduler:

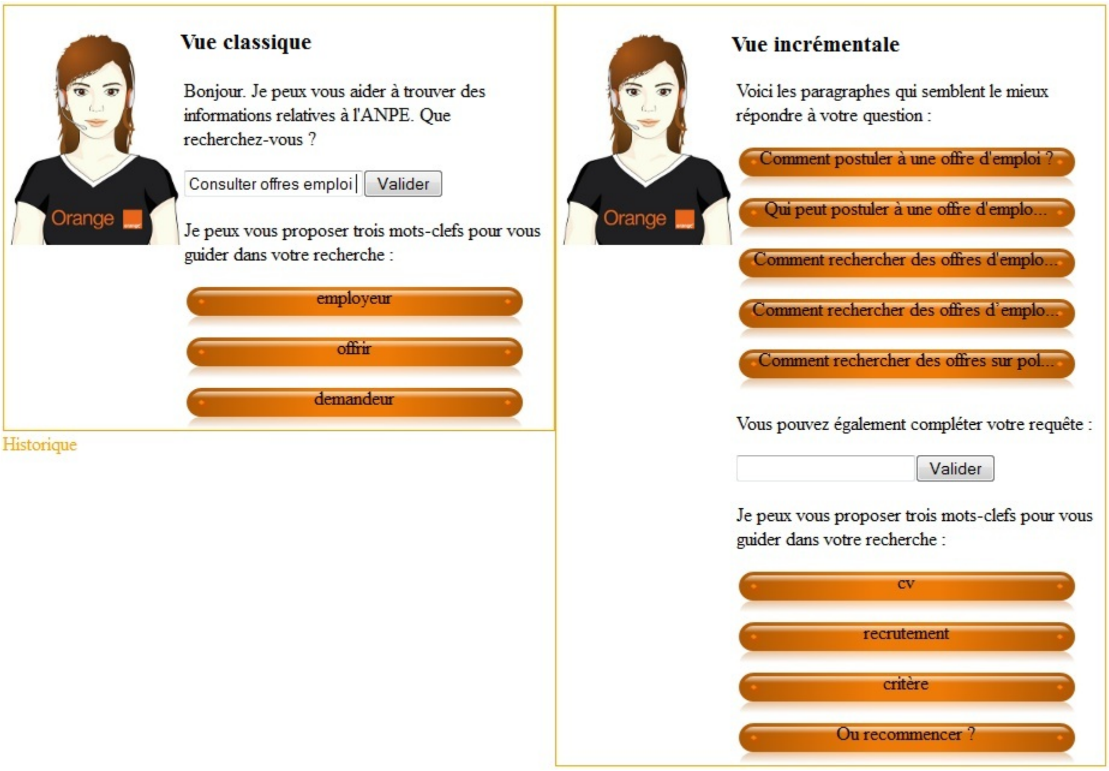
- **Commit:** The Scheduler commits to a partial request and the corresponding response when it decides to deliver the latter to the client, hence taking the floor immediately and not waiting for any further information. In that case, the simulated context is saved into the real context.

- **Cancel:** The scheduler cancels the context changes when it decides to discard the very last response obtained from the service. In that case, the real context is copied into the simulated one, rollbacking it to its original state. As shown in Fig 4.3, this decision is only made when a new - potentially more complete - partial request is received from the client.

The way the real and the simulated context are managed through the commit and the cancel actions is illustrated in Fig. 4.4.

4.2 Illustration

4.2.1 A textual dialogue system: CFAsT


Orange
Orange
Valider
employeur
offrir
demandeur
Historique
Comment postuler à une offre d'emploi ?
Qui peut postuler à une offre d'emplo...
Comment rechercher des offres d'emplo...
Comment rechercher des offres d'emplo...
Comment rechercher des offres sur pol...
Valider
cv
recrutement
critère
Ou recommencer ?

Vue classique

Bonjour. Je peux vous aider à trouver des informations relatives à l'ANPE. Que recherchez-vous ?

Consulter offres emploi

Je peux vous proposer trois mots-clefs pour vous guider dans votre recherche :

Vue incrémentale

Voici les paragraphes qui semblent le mieux répondre à votre question :

Vous pouvez également compléter votre requête :

Je peux vous proposer trois mots-clefs pour vous guider dans votre recherche :

Fig. 4.5 The incremental version of the CFAsT project

CFAsT stands for Content Finder AssistanT. This application developed at Orange Labs is aimed to automatically generate a virutal assistant that helps the user to efficiently find a specific content in a database. At each dialogue turn, the user provides some new information about his target and by using a keyword spotting algorithm, the system keeps narrowing the set of possibilites. The interface is made of a text box with a *validate* button. The dialogue service is deployed as a web service on a servlet container and the client is a javascript web page loaded on the user's browser.

In order to make it incremental, another servlet has been deployed on the same container as the service and the javascript requests have been re-targeted toward it (the service is no longer directly visible by the client). Moreover, as discussed earlier, the dialogue context in the service have been duplicated. In this setup, *EndTurnCond* corresponds to the event of clicking on *validate* (or hitting the carriage return). A word-based micro-turn configuration has been implemented, therefore, *EndMicroTurnCond* is the action of hitting the space bar. *ServiceReqCond* is not useful here, hence, it is true all the time.

This implementation is aimed to provide a first proof of concept of the methodology introduced here. However, $signal_{ETC}$ = *CommitCond* = *EndTurnCond* as the Scheduler never takes commit decisions by itself. Therefore, this is a very limited illustration.

4.2.2 A spoken dialogue system: Dictanum

Number dictation is an activity that can be encountered in several kinds of tasks: phone number dictation, credit card...Analysing human conversations is such a situation shows that the number at hand is often broken into small pieces (a few digits each) and a confirmation follows each one of them. Therefore, the number is incrementally grounded so that in the case of a misunderstanding, only the last part dictated has to be repeated. Traditional dialogue systems behave differently as they ask the user to enter the whole number at once. This translates into long and tedious dialogues as only one little mistake is enough to make the user repeat the number from the beginning, often for several times. For example:

SYSTEM: Welcome to the Dictanum service. Please dictate your number.
USER: 01 45 65 79 98.
SYSTEM: The dictated number is 01 45 67 79 98. Is that correct?
USER: No.
SYSTEM: Sorry. Let us try again. Please dictate your number.
USER: 01 45 65 79 98.
SYSTEM: The dictated number is 01 47 65 79 98. Is that correct?
USER: No.
SYSTEM: Sorry. Let us try again. Please dictate your number.
[...]
SYSTEM: The dictated number is 01 45 65 79 98. Is that correct?
USER: Yes.

SYSTEM: Thank you for using Dictanum. Bye.

With incremental dialogue processing, it is possible to design a strategy that is closer to the human behaviour. To show that, a number dictation dialogue system called Dictanum has been implemented in our lab. It performs turn-based dialogue hence asking the user for the whole number at once. Here, the client is a web page displayed on the browser that uses Google API for both the ASR and the TTS tasks and the service is servlet deployed on a Tomcat container.

Afterwards, the approach introduced here has been used to build the incremental version of the system (like in the case of the CFAsT application, the Scheduler has been deployed as a servlet on the same container as the service). To do so, two silence duration thresholds have been defined: the short silence threshold δ_s and the long one Δ_s. *EndMicroTurnCond* is triggered when a short silence is detected and similarly, *EndTurnCond* corresponds to long silences. The system has been modified to detect these short silences and to deliver a feedback (repeating the last 4 digits) if detected. If the user ignores the feedback and keeps dictating his number, the system keeps on adding digits to his list, however, if the user starts his next utterance with *No*, the feedback content is deleted from the number. Here is a dialogue example:

SYSTEM: Welcome to the Dictanum service. Please dictate your number.
USER: 01 45
SYSTEM: 01 45
USER: 65 79
SYSTEM: 67 79
USER: No, 65 79
SYSTEM: Sorry, 65 79
USER: 98
SYSTEM: 98
USER: ...
SYSTEM: The dictated number is 01 45 65 79 98. Is that correct?
USER: Yes.
SYSTEM: Thank you for using Dictanum. Bye.

Dictanum also offers the possibility for the user to interrupt the system during the final feedback, in order to make local corrections. To do that, this feedback is sent to the TTS in the following format: *The dictated number is 01 <sep> 45 <sep> 65 <sep> 79 <sep> 98. Is that correct?*. The latter pronounces the sentence chunk after chunk (chunks are delimited using the separator <*sep*>), each chunk lasting for the same number of micro-turns. This leads to the following kind of strategy:

SYSTEM: The dictated number is: 01 45 67...
USER: No, 65.
SYSTEM: Sorry. The dictated number is 01 45 65 79 98. Is that right?
USER: Yes.
SYSTEM: Thank you for using Dictanum. Bye.

4.3 Discussion

4.3.1 Levels of incrementality

Dialogue systems can be classified in four categories given the way they integrate incremental behaviour. The first category is made of traditional systems [19]. Then comes the second category where traditional systems locally simulate a few incremental behaviours. For instance, in [10], the system enumerates a list of options and the user selects the one that fits him best by uttering *Yes* or *Ok* for example (REF_RAW in the taxonomy introduced in Chap. 3). The architecture introduced in this thesis belongs to the third category where incremental behaviour is obtained based on modules that are innately non-incremental (the service in our case). Other examples are described in [37] and [14]. Finally, the fourth category is made of incremental dialogue systems that are constituted of fully-incremental modules. In [33], an abstract model for incremental architectures is presented where all the categories can fit, but the work that has been pursued by the authors and their research groups later on goes along with the spirit of this last category.

Categories 2, 3 and 4 embed different features related to incremental behaviour as summarised in Fig. 4.1.

4.3.2 Enhancing a traditional dialogue system's turn-taking abilities at a low cost

4.3.3 Separating dialogue management from floor management

Features	**Category 1**	**Category 2**	**Category 3**	**Category 4**
TTS interruption after input analysis	-	+	+	+
Link interruption time with TTS	-	+	+	+
User interruption by the system	-	-	+	+
Better reactivity	-	-	+	+
Optimal processing cost	-	-	-	+

Table 4.1 Available features for dialogue systems given the way they integrate incrementality

Chapter 5

Incremental dialogue simulation

5.1 Agenda management task

A personal agenda assistant has been implemented as our task for the experiments. The user can add, move or delete events in his agenda. For instance, a request could be: *I would like to add the event football game on March* 3rd *from 9 to 10 pm*[1]. This is a slot filling task with four slots:

- **ACTION:** The type of action the user wants to perform. Can take three different values: ADD, MODIFY or DELETE.
- **DESCRIPTION:** The title of the event.
- **DATE:** The date of the event.
- **SLOT:** The time slot of the event.

However, the no overlap is tolerated between events in the agenda.

The US is given two lists of events: *InitList* and *ToAddList*. The first one contains the events that already exist in the dialogue before the dialogue and the second one contains the ones that the US is supposed to add during the dialogue. Each event is associated with a priority value and the US must prefer adding the ones with high probability first. Its aim is to make as many events as possible fit in the agenda.

---

[1] The dialogues are actually in french but they are translated in English in this thesis

Table 5.1 NLU rules types

Rule type	Description	Example
Tag	Words are associated to a label	remove : [DELETE]
Regular expressions	Spots words that satisfy a regular expression	Regex([0-9]+) : NUMBER($word)
Combine	Words and concepts are mapped into a new concept	Combine(NUMBER,MONTH) : DATE

5.2 The service

5.2.1 Natural Language Understanding

A recursive algorithm transforms the user's utterance hypothesis into a concept tree. To do that, a set of rules have been defined. Each rule transforms a word, a concept or any combination of the two into a new concept. Three types of rules are used; they are depicted in table 5.1

The NLU algorithm keeps applying these rules the order they appear in the rules file until two consecutive results are identical. For instance, parsing the sentence *I want to add the event birthday party on January* 6th *from 9pm to 11pm* is performed following these steps:

1. **I want to ADD the TAG_EVENT birthday party on MONTH(January) NUMBER(6) from TIME(9,0) to TIME(11,0)**
    - add : [ADD]
    - event : [TAG_EVENT]
    - Regex(janvier|...|decembre) : MONTH($word)
    - Regex([0-9]+) : NUMBER($word)
    - Regex((([0-1]?[0-9])|(2[0-3]))h([0-5][0-9])?) : TIME($word)
2. **I want to ADD the TAG_EVENT birthday party on DATE(6,1) SLOT(TIME(21,0),TIME(23,0))**
    - Combine(NUMBER,MONTH) : DATE(NUMBER,MONTH)
    - Combine(from,TIME_1,to,TIME_2) : SLOT(TIME_1,TIME_2)
3. **I want to ADD EVENT(birthday party, DATE(6,1), SLOT(TIME(21,0),TIME(23,0)))**
    - Combine(TAG_EVENT,$x,on,DATE,SLOT) : EVENT($x,DATE,SLOT)
4. **I want ACTION(ADD, EVENT(birthday party, DATE(6,1), SLOT(TIME(21,0),TIME(23,0))))**

- Combine(ADD,EVENT) : ACTION(ADD,EVENT)

5.3 User simulator

5.3.1 Overview

The User Simulator (US) is composed of six modules (see Figure 5.1). The Intent Manager, the NLU, the Verbosity Manager, the ASR output simulator, the NLG and the Patience Manager. In every incremental dialogue setup, each dialogue turn is divided into smaller micro-turn given a particular criteria; a micro-turn could be a time window (500 milliseconds for example), it could also be word-based or concept-based (a new word or a new concept marks the end of a micro-turn). The US described here is word-based.

At each micro-turn, the US generates a partial N-Best. It embeds the whole utterance from the beginning of the turn (*restart incremental* mode [33]). On the other hand, either the US receives an answer from the Scheduler at a certain micro-turn and it stops speaking, either it does not and it continues speaking if it has additional thing to say, or spontaneously releases the floor. When the dialogue lasts for too long without achieving the task at hand, the US can end the dialogue.

In the following, the different components of the US are described in detail.

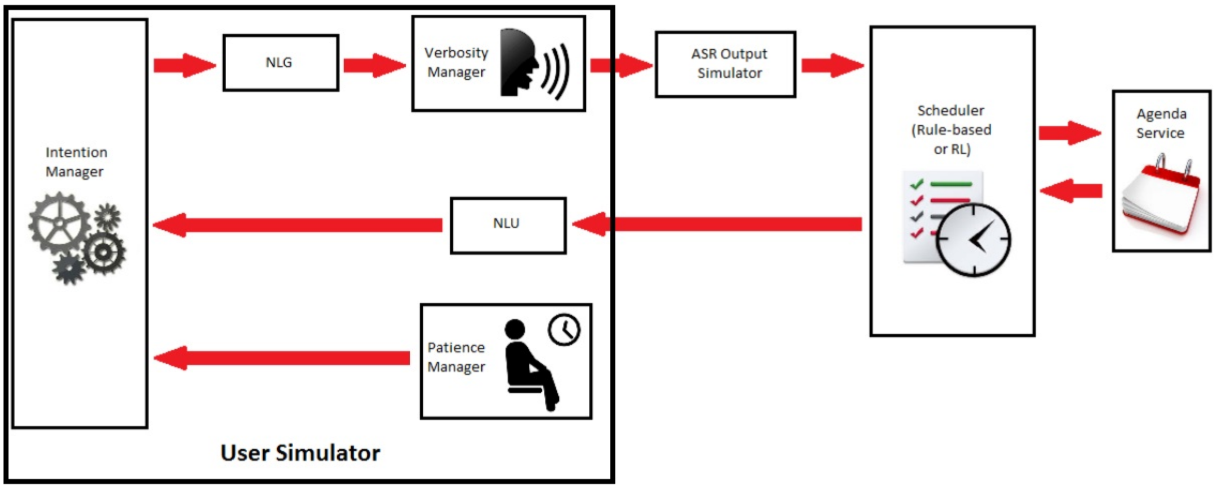


Fig. 5.1 Simulated environment architecture

5.3.2 Intent Manager

The aim of the US is to make the biggest number possible of events taken from *InitList* and *ToAddList* fit in the agenda. The events with the highest priority are considered first. To do so, the Intent Manager runs Algorithm 1 where

- **actionList:** the list of actions to be performed by the system (initially the list of ADD actions corresponding to *ToAddList*).
- **act():** function that performs a list of actions.
- **perform():** function that performs a single action.
- **alternatives():** function that returns the alternative events of the one corresponding to the input action.
- **conflictsByAlt:** contains couples (event,conflictSet) mapping events to the sets of conflicting events in the agenda.
- **maxPrioByAlt:** contains couples (event, maxPrio) mapping events to the maximum priority in conflictsByAlt(event).

5.3.3 NLG and Verbosity Manager

The NLG module transforms action concepts as well as date, slot, description and yes/no concepts into simple straight forward utterances. For instance the NLG result for the intent ACTION(ADD, EVENT(birthday party, DATE(6,1), SLOT(TIME(21,0),TIME(23,0)))) is *add the event birthday party on January* 6th *from 9pm to 11pm.*

Nevertheless, compared to human/human conversations, limiting interactions to this kind of simple utterances is not realistic. Therefore, they are enhanced in the Verbosity Manager with prefixes like *I would like to*, *Is it possible to*...and suffixes like *if possible*, *please*... In [12], a corpus study showed that users tend to go off-domain and to repeat the same information several times in the same sentence. These behaviours are also replicated in the Verbosity Manager: 10% of the times, the US utters an off-domain sentence, and 30% of the times where there is a misunderstanding, it repeats the same information twice. These proportions have been estimated based on the corpus study mentioned before.

```
Algorithm act(actionList)
    if actionList not empty then
        added ← perform(actionList(0))
        if not added then
            alt ← alternatives(actionList(0))
            conflictsByAlt ← []
            maxPrioByAlt ← []
            addedAlt ← false
            while alt not empty AND not addedAlt do
                addedAlt ← perform(alt(0))
                if not addedAlt then
                    conflictsByAlt ← [conflictsByAlt (alt(0),conflicts(alt(0)))]
                    maxPrioByAlt ← [maxPrioByAlt (alt(0),max priority of
                    conflicts(alt(0)))]
                    alt.remove(0)
                end
            end
            if not addedAlt then
                selectedAlt ← argmin maxPrioByAlt
                movedToAlt ← true
                while conflictsByAlt no empty AND movedToAlt do
                    moveActionList ← list of move actions to the different alternatives
                    of conflict
                    movedToAlt ← act(moveActionList)
                end
                if movedToAlt then
                    perform(ationList(0))
                else
                    return false
                end
            end
        end
        actionList.remove(0)
        return act(actionList)
    end
    return true
```

Algorithm 1: Intent Manager algorithm

5.3.4 ASR output simulator

The ASR output simulator generates an N-Best that is updated at each new micro-turn. For instance, if at a certain point, the US uttered *I would like to add the event birthday party on...*, a possible N-Best could be (the numbers between brackets represent ASR scores):

- (0.82) I would like to add the event birthday party on
- (0.65) I like to add the event birthday party on
- (0.43) I have had the event birthday party
- (0.33) I would like to add the holiday party
- (0.31) I like to add the holiday party on

More formally, at time t, the N-Best is an N-uplet $(s_1^{(t)}, hyp_1^{(t)}), ..., (s_N^{(t)}, hyp_N^{(t)})$. At time t+1, a new word w_{t+1} is send to the ASR output simulator and the latter calculates a new associated N-Best. Therefore, at this stage, the system has two N-Best (one for the whole current partial utterance and one for the new word) that it has to combine to obtain the new partial utterance N-Best. In the following how the word N-Best is calculated and how it is incorporated in the partial utterance N-Best.

A Word Error Rate (WER) is given as a parameter to the ASR output simulator. It controls the noise level that one wants to simulate. A Scrambler is used to generate a modified version of some selected words. It can achieve three types of modifications:

- Replace the word with a different word taken randomly from a dictionary (probability: 0.7).
- Add a new word (probability : 0.15).
- Delete the word (probability : 0.15)

Starting from here, we describe the algorithm used to generate the N-Best associated with a single word.

1. Determine whether w_{t+1} is among the N-Best or not with a probability that is computed as follows: (1 - WER) + INBF.WER, where INBF (In N-Best Factor) is a parameter between 0 and 1 (set to 0.7 in this thesis). If w_{t+1} is not in the N-Best, then the latter contains only scrambled versions of this word and we directly jump to step 4.

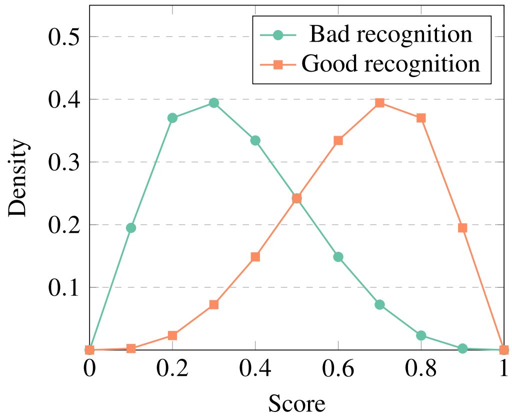


Fig. 5.2 ASR score sampling distribution

2. The first hypothesis is set to be w_{t+1} with a probability of (1-WER), otherwise, it is a scrambled version of it.
3. If the the first hypothesis is not w_{t+1}, then this word's position is randomly chosen between 2 and N. Moreover, the other hypothesis are scrambled versions of it.
4. The confidence score associated to the best hypothesis (s_0) is sampled as sigmoid(X) where X is a gaussian. The distribution mean is -1 if the first hypothesis is wrong and 1 when it is right. The standard deviation of X is one. By taking the sigmoid, this leads to two distributions (depicted in Figure 5.2) with a mean of 0.3 and 0.7 and a standard deviation of 0.18 for both (which can be changed to simulate different levels of accuracy of the confidence score model. The closest to 0, the better the model).
5. The scores for the other hypotheses are computed in an iterative way. For i between 2 and N, s_i is uniformly sampled in $[0, s_{i-1}]$.

In incremental settings, the ASR input is a continuous audio stream. In reality, it is split into small consecutive chunks of data. At each new input arrival, the output is updated. However, a new chunk of audio data in the input does not necessarily translate into a new chunk of text added to the output. In reality, the latter can be partially or even completely changed while update. An example from [33] is when the user utters the word *forty*: the system first understands *four* and then *forty*. The transition between the two hypotheses does not translate into an addition of information but the first hypothesis has to be revoked and replaced by the second one. This phenomenon is referred to as the *ASR instability*.

To replicate this behaviour, a language model is needed to compute the scores corresponding to the different hypotheses in the N-Best. Therefore, for sentences that are more in alignment with the model will have high scores thus pushed to the top of this N-Best. Here, we use the NLU knowledge as a proxy for the language model by making the following assumption: *the more an utterance generates key concepts once fed to the NLU, the more it is likely to be the correct one*. Therefore, as soon as a new action, date or time slot concept is detected in hyp_i, s_i is boosted as follows:

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

where BF is the Boost Factor parameter. Here it is set to 0.2. An illustration of this mechanism is provided in Fig. 5.3.

5.3.5 Timing and patience manager

When it comes to incremental processing, timing is key. However, the main objective of simulation is to generate dialogues in an accelerated mode, hence, no track of timing is kept. In order to approximate durations, the user's and the system's speech rates are considered to be 200 words per minute [45].

Users tend to get impatient, at various degrees, when dialogue systems take too long to accomplish the task they are asked for. To simulate this behaviour, a duration threshold is chosen at each new dialogue task (adding, modifying of deleting an event) that will cause the user to hangup when reached. It is computed as follows

$$d_{pat} = 2\mu_{pat}.sigmoid(X) \tag{5.1}$$

where X follows a gaussian distribution of mean 0 and variance 1 and consequently μ_{pat} is the mean duration threshold.

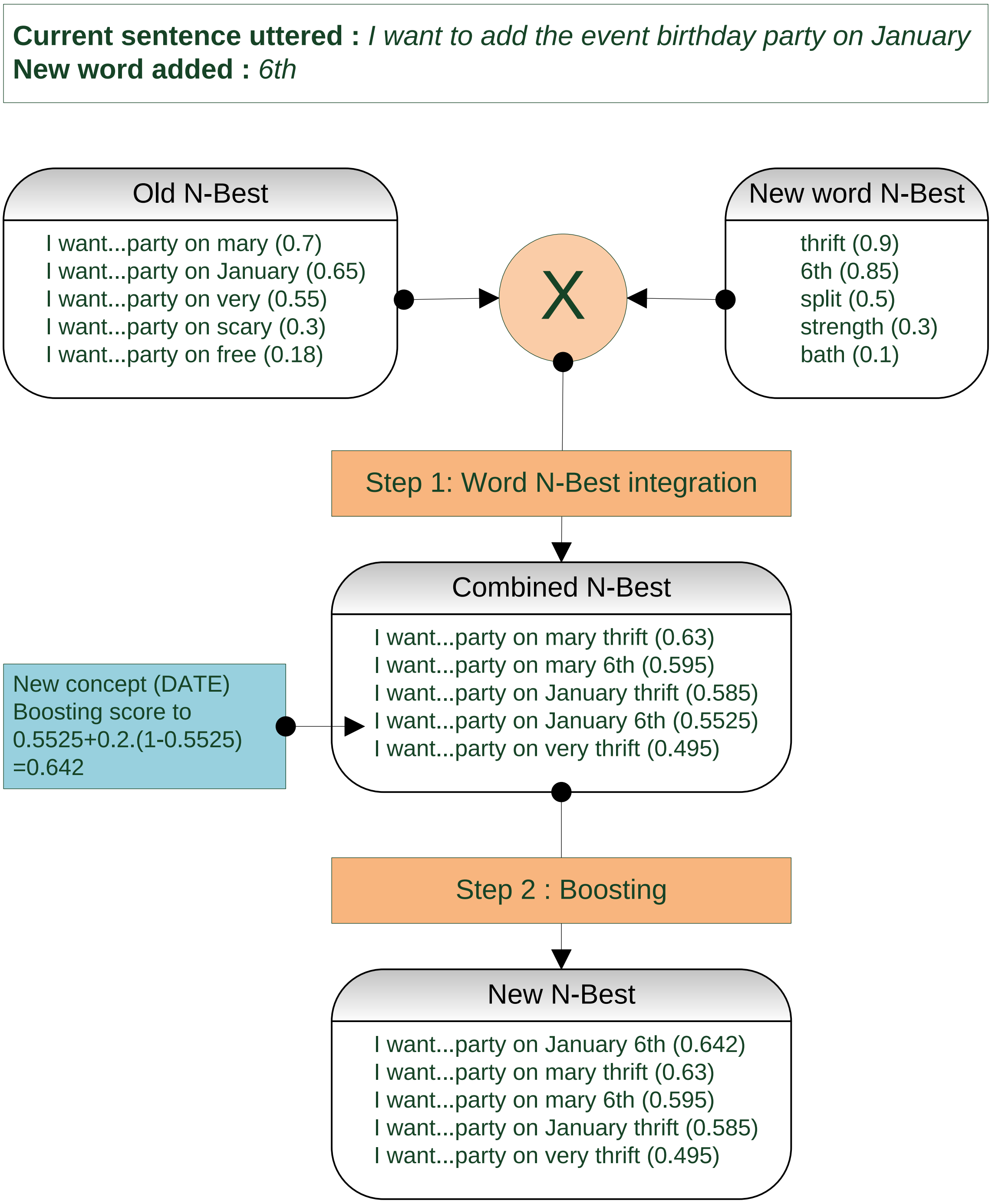

Current sentence uttered : I want to add the event birthday party on January
New word added : 6th
Old N-Best
I want...party on mary (0.7)
I want...party on January (0.65)
I want...party on very (0.55)
I want...party on scary (0.3)
I want...party on free (0.18)
X
New word N-Best
thrift (0.9)
6th (0.85)
split (0.5)
strength (0.3)
bath (0.1)
Step 1: Word N-Best integration
Combined N-Best
I want...party on mary thrift (0.63)
I want...party on mary 6th (0.595)
I want...party on January thrift (0.585)
I want...party on January 6th (0.5525)
I want...party on very thrift (0.495)
New concept (DATE)
Boosting score to
0.5525+0.2.(1-0.5525)
=0.642
Step 2 : Boosting
New N-Best
I want...party on January 6th (0.642)
I want...party on mary thrift (0.63)
I want...party on mary 6th (0.595)
I want...party on January thrift (0.585)
I want...party on very thrift (0.495)

Fig. 5.3 An illustration of the incremental ASR output N-Best update

Chapter 6

Handcrafted strategies for improving dialogue efficiency

6.1 User and system initiative

6.1.1 Strategies

Most of the current task oriented dialogue systems operate in a slot-filling manner. Here, we identify three strategies to fill all the slots needed by the system:

- **System Initiative (SysIni):** The system asks the user for the different slot values one by one, one at each dialogue turn like in the following example:

  SYSTEM: What kind of action do you want to perform?

  USER: Add.

  SYSTEM: Please specify a title.

  USER: Dentist.

  SYSTEM: Please specify a date.

  USER: March <noise>.

  SYSTEM: Sorry I don't understand.

  USER: March 7th.

  SYSTEM: Please specify a time slot.

  USER: From 10am to 11am.

SYSTEM: Ok. So you want to add the event dentist on March 7th from 10am to 11am. Is that right?

USER: Yes.

This strategy is very tedious, however, it has the advantage of baby-stepping the dialogue and recovering the errors progressively.

- **User Initiative (UsrIni):** The user is asked by the system to make a full sentence with all the information needed to process the request. For example:

SYSTEM: What can I do for you?

USER: I would like to add the event dentist on March <noise> from 10am to 11am.

SYSTEM: Sorry I don't understand.

USER: I would like to add the event dentist on March 7th from <noise> to 11am.

SYSTEM: Sorry I don't understand.

USER: I would like to add the event dentist on March 7th from 10am to 11am.

SYSTEM: Ok. So you want to add the event dentist on March 7th from 10am to 11am. Is that right?

USER: Yes.

With this strategy, the dialogue can progress much faster, nevertheless, unlike SysIni, when there is still missing information (even one slot), the user has to repeat all his request.

- **Mixed Initiative (MixIni):** Just like UsrIni, the system asks for all the slots at once but if there is still missing information, it switches to SysIni in order to gather the missing slot values one by one. For example:

SYSTEM: What can I do for you?

USER: I would like to add the event dentist on March <noise> from 10am to 11am.

SYSTEM: Please specify a date.

USER: <noise>.

SYSTEM: Sorry I don't understand.

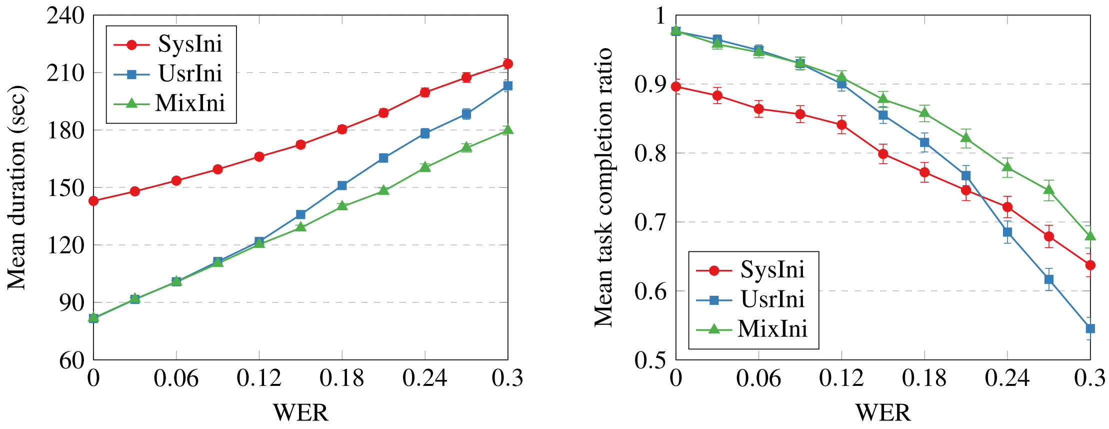


Fig. 6.1 Simulated mean duration (left) and dialogue task completion (right) for different noise levels

USER: March 7th.

SYSTEM: Ok. So you want to add the event dentist on March 7th from 10am to 11am. Is that right?

USER: Yes.

By elaborating such a hybrid strategy, we hope that it will ally both advantages of SysIni and UsrIni. In the following, we use the simulator to investigate the performance of these strategies under different conditions of noise.

6.1.2 Experiments

For the experiments, three dialogue scenarios have been used. As described in 5.3.2, a scenario is specified by two lists of events: *InitList* and *ToAddList*. The lists corresponding to the three scenarios are given in Table 6.1.

We vary the WER between 0 and 0.3 with a step of 0.03 in order to analyse the effect of noise on the different strategies. For each noise level, the three scenarios and run 1000 times. The mean duration of the dialogues, their task completion as well as the corresponding 95% confidence intervals are depicted in Fig. 6.1.

In low noise setups, SysIni is clearly less efficient as UsrIni; the dialogues take twice more time to finish with a lower completion rate (actually, it is a consequence of longer dialogue durations given the way the Patience Manager is designed). Nevertheless when the noise level reaches about 0.2, SysIni gives offers better completion rates. The duration is still lower in spite of the correlation between the two metrics. This is due to the fact that the durations distribution for UsrIni is centered on short dialogues whereas the distribution for

Table 6.1 *The three scenarios used in the simulation*

Scenario	Title (Priority)	Date	Slot	DateAlt1	SlotAlt1	DateAlt2	SlotAlt2
1-Init	Guitar lesson(4)	November 17th	14:00-15:30	November 15th	9:45-11:15		
1-ToAdd	Book reading(8)	November 19th	10:30-12:30	November 14th	9:30-11:30	November 18th	16:30-18:30
	Watch the lord of the rings(12)	November 13th	9:30-12:30	November 15th	11:15-14:15		
2-Init	Guitar lesson(4)	November 17th	14:00-15:30	November 15th	9:45-11:15		
2-ToAdd	Tennis(5)	November 17th	13:15-15:15	November 19th	15:15-17:15		
	Gardening(9)	November 18th	13:15-15:15	November 14th	12:30-14:30		
3-Init	Guitar lesson(4)	November 17th	14:00-15:30	November 15th	9:45-11:15		
	Holidays preparation(1)	November 16th	12:30-14:30	November 17th	12:15-14:15		
	House cleaning(6)	November 13th	14:15-16:15	November 17th	15:30-17:30		
3-ToAdd	Give back book(7)	November 16th	14:00-14:30	November 13th	14:00-14:30		

SysIni is centered on average ones. Finally, MixIni seems to be the best strategy as it allies both the advantages of UsrIni and SysIni.

6.2 Incremental strategies

In this study, the two metrics under focus are duration and task completion (used as a proxy to measure dialogue efficiency). Nevertheless, one of the main motivations behind incremental dialogue process is to increase human likeness and the dialogue fluency by achieving smooth turn-taking. Some TTPs from the taxonomy established in Chapter 3 are more likely to improve these last aspects instead of the metrics that interest us most directly, like the BACKCHANNEL for instance. Therefore, only a few TTPs from the taxonomy established in Chapter 3 have been implemented: those that are more likely to improve our two objective metrics. However, the phenomena that are not studied here could have an indirect impact on duration and task completion as objective and subjective metrics are somehow correlated. In simulation, this is not visible though.

INIT_DIALOGUE and END_POINT are two phenomena that already exist in traditional dialogue systems. In the Scheduler module (see Chapter 4), we add the following phenomena: FAIL_RAW, INCOHERENCE_INTERP, FEEDBACK, BARGE_IN_RESP. The last phenomena have also been implemented in the US to make it able to interrupt the system. Section 6.2.2 provides the implementation details of these TTPs.

6.2.1 ASR instability

Because of the ASR instability phenomenon explained in 5.3.4, the current partial utterance is not guaranteed to be a prefix of future partial utterances and it is likely to change. However, the words at the end of this partial utterance are more likely to change than the ones at the start. In [24], it is shown that words that lasted for more than 0.6 seconds have 90% chance of staying unchanged. In this work, the speech rate (user and system) is supposed to be

200 words per seconds [45] so 0.6 seconds corresponds to two words. Based on that, the handcrafted TTP implementation takes decisions by looking at the current partial hypothesis without its last two words (called the *the last stable utterance*). This margin is called the *Stability Margin* (SM = 2).

6.2.2 TTP implementation

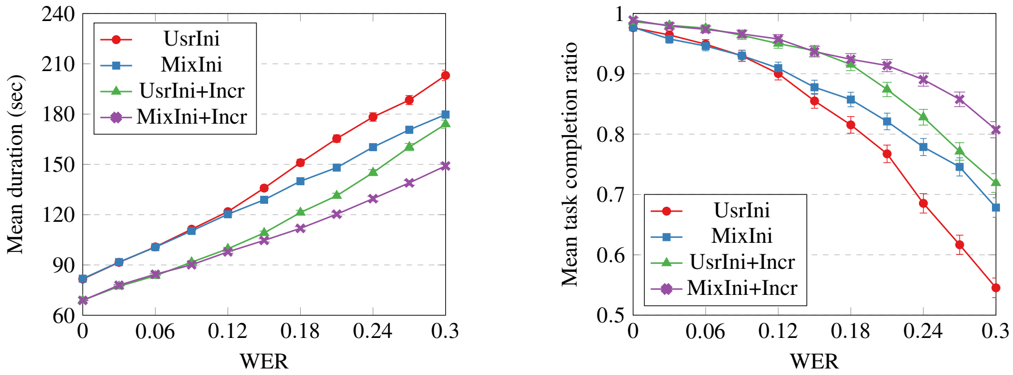
In the following, the TTP implementation is described. The settings and the parameter values are handcrafted given our experience. In the following chapters, we will try to optimise them from data.

FAIL_RAW: Happens when the user speaks for two long with no key concept detected in her speech. It depends on the system's last question (the type of information it is waiting for). Here, it is triggered when the system asks an open question (all the slots at once) and the user's utterance contains 6 words with no action type detected (add, modify or delete). This threshold is set to 3 in the case of yes/no questions, 4 for date questions and 6 for slot questions.

INCOHERENCE_INTERP: The system can fully understand the user's partial utterance and yet have a good reason to barge-in. This is the case when the user injects new information that is not coherent with the dialogue context. For instance, if the user says *I want to move the event football game to January...* and there is no event football game in the agenda, then letting him finish his utterance is a waste of time. As a consequence, it makes sense for the system to barge-in. In this implementation, this event is triggered as soon as the last stable utterance generates an overlap with an existing event in the agenda, or it tries to move or delete a non existing event.

FEEDBACK: This phenomenon translates into repeating a part if not the whole user's utterance. However, for the sake of simplicity, we limit ourselves here to the case where only the last word of the last stable utterance is repeated. The moments when feedbacks are performed in this implementation are the moments when the ASR confidence score drops below a certain threshold. To isolate the confidence score corresponding to the target word only, the ratio s_{t-SM}/s_{t-SM-1} is computed (as the confidence score of a sentence is the product of the scores associated with the words in this sentence). Here, this event is triggered whenever this ratio drops under 0.7.

Fig. 6.2 Mean dialogue duration and task completion for generic strategies.



BARGE_IN_RESP (System): Depending on the last system dialogue act (apart from dialogue acts reporting errors), the system can choose to barge-in once it has all the information needed to provide an answer. Again, it should also wait for the SM.

BARGE_IN_RESP (User): When the user gets familiar with the system, it tends to predict the system's dialogue act before the system finishes its sentence. Unlike the previous phenomena, this one is due the a user's decision. Hence, it has been implemented in the US.

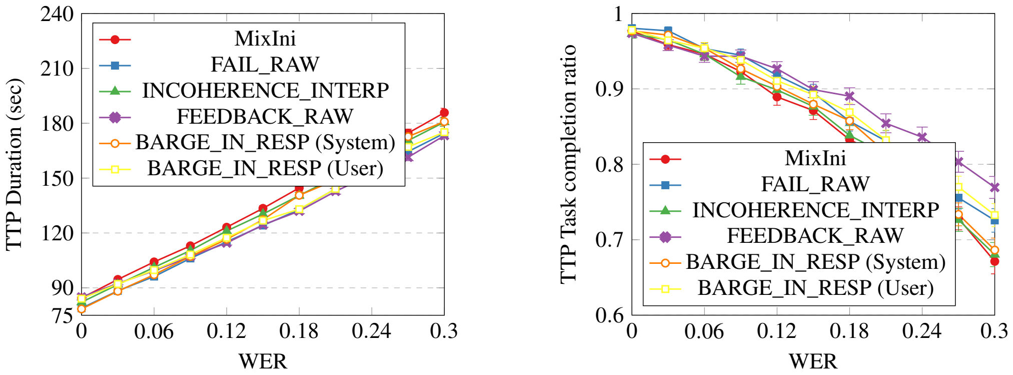
6.2.3 Experiments

In this experiment, the TTPs phenomena described in Sect. 6.2.2 are used on the top of the slot-filling strategies introduced in Sect. 6.1.1. Incremental processing is useful when the user makes long utterances. This is not the case in the SysIni strategy where her utterances are very short. Therefore, incremental behaviour have only been added to UsrIni and MixIni to form two new strategies: UsrIni+Incr and MixIni+Incr. The associated performances are depicted in Fig. 6.2.

Adding mixed initiative behaviour or incrementality to UsrIni are both ways to improve its robustness to errors. Fig. 6.2 shows that in our case, incrementality is more efficient. Most importantly, it is shown here that MixIni and incremental behaviour can be combined to form the best strategy.

As already mentioned in Chapter 3, the main objective of our TTP taxonomy is to break human dialogue turn-taking into small pieces, and hopefully get a better understanding of it. To illustrate this approach, we take a deeper look at MixIni+Incr by isolating its different components: FAIL_RAW, INCOHERENCE_INTERP, FEEDBACK, BARGE_IN_RESP (User) and BARGE_IN_RESP (System). The results reported in Fig. 6.3 show that FEEDBACK contributes the most to improve the baseline followed by BARGE_IN_RESP (User)

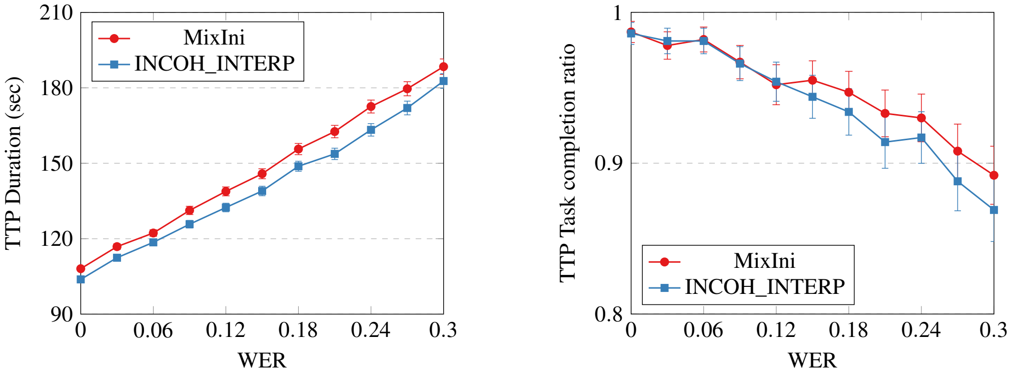
Fig. 6.3 Mean dialogue duration and task completion for different turn-taking phenomena.



and FAIL_RAW. INCOHERENCE_INTERP and BARGE_IN_RESP (System) seem to have no effect. This is due to the fact that in general, to detect an incoherence, one must wait until the end of the utterance (same requirement for detecting all the information needed to barge-in and provide an answer). One might argue that in some case, the US tries to refer to a non existing event (in the case of a MODIFY or DELETE action), therefore triggering an incoherence. In reality, our service is able to recognise an existing event even if only a prefix of its title is recognised. As a consequence, INCOHERENCE_INTERP is rarely triggered in the middle of a request. In order to force that case to occur more often, a scenario where the US has to try to move an event five times before encountering a free slot has been designed. The results are shown in Fig. 6.4: this time, INCOHERENCE_INTERP has a true added value.

This little experiment raises an important point when it comes to studying efficiency in task oriented dialogues and as far as turn-taking mechanisms are concerned. The nature of dialogue is very diverse, therefore, results and the following conclusions (whatever the chosen metric is) should not be given separately from the dialogue frame. Here, the focus is efficiency, however, human-machine dialogue can also be used in a non task oriented fashion. The metrics chosen here make no longer pertinent in such a frame and it would make more sense to consider subjective ones. Moreover, even in task oriented dialogue, booking a train is very different from dictating a text message. In the first case, a keyword spotting NLU is adapted to the task at hand but in the second situation, the system does try to understand what the user is saying. The feedback can be very useful in this situation for instance.

Fig. 6.4 INCOHERENCE_INTERP evaluated in a more adapted task



Chapter 7

Reinforcement learning for turn-taking optimisation

7.1 Model

7.1.1 State representation

The following features are used to describe the system state:

- **SYSTEM_REQ:** At each dialogue turn, the system is requiring a particular information. For instance, after an open question, it is waiting for all the slot values to be provided at once but it can also be waiting for a specific question or a response to a confirmation. This feature refers to the information that it is waiting for at this moment. It can take 6 different values.

- **LAST_INCR_RESP:** As described in Chap. 4, the Scheduler stores the last response it gets from the service at each micro-turn. We use that as our second feature. 11 different values can be taken by this feature.

- **NB_USER_WORDS:** This feature is a counter of the number of words since the last change of LAST_INCR_RESP (the number of words since the service did not change its mind about the response to deliver).

- **NORMALISED_SCORE:** At each micro-turn, the ASR score is updated: it is multiplied by the ASR corresponding to the new incoming word (see Fig. 5.3). Except from the cases where a boost takes place, the score keeps decreasing as the user speaks. To avoid penalising long sentences, the score is normalised by taking the geometric mean over the words (of course, this is an approximation as the number of inputs that formed the current ASR hypothesis is not exactly the number of words because of deletions and additions in the Scrambler). If s is the current score for n number of words, NORMALISED_SCORE = $s^{\frac{1}{n}}$.

- **TIME:** Corresponds to the duration of the current episode.

A linear model is used to represent the Q-function. First, we noticed that 21 combinations between SYSTEM_REQ and LAST_INCR_RESP are frequently visited (the others barely happen or not at all). Therefore, 21 features are defined δ_1,...,δ_{21} where $\delta_i = 1$ if and only if the current state corresponds to the ith combination, and 0 otherwise. The rare combinations are not included in the model for two reasons. First, they require maintaining heavier models for with no real improvements over the simpler ones and second, Fitted-Q algorithm was chosen for resolving the MDP and it involves a matrix inversion that should be well conditioned. Having features that are equal to zero most of the time is not compliant with that condition.

NB_USER_WORDS is represented by three RBF functions ϕ_1^{nw}, ϕ_2^{nw} and ϕ_3^{nw} centered in 0, 5 and 10 with a standard deviation of 2, 3 and 3. In other words

$$\phi_i^{nw} = \exp(\frac{NB\_USER\_WORDS - \mu_i}{2\sigma_i^2})$$

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

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

Similarly, NORMALISED_SCORE is also represented using two RBF functions ϕ_1^{ns} and ϕ_2^{ns} centered in 0.25 and 0.75 and with a standard deviation of 0.3 for both.

Finally, TIME is normalised so that it is near zero at the beginning of the episode and around 1 when the duration reaches 6 minutes (the maximum duration due to patience limit):

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

There is no need to use RBFs for this last feature as the Q-function is supposed to be monotonous with respect to it. The more the dialogue last, the more likely the user would hang up.

Therefore, the dialogue state is represented by the following vector

$$\Phi(s) = [1, \delta_1, \delta_2, \delta_3, \phi_1^{nw}, \phi_2^{nw}, \phi_3^{nw}, \phi_1^{ns}, \phi_2^{ns}, T]$$

7.1.2 Actions, rewards and episodes

The system can perform the action WAIT and the action SPEAK (see Chap. 6 for a description of these actions). The action REPEAT introduces more complexity to the system and therefore, it is left for future work.

At each micro-turn, the system receives a reward $-\Delta t$ which correspond to the opposite of the time elapsed since the micro-turn before. Moreover, there are two rewarding situations, where the system gets a reward of 150:

- The Scheduler retrieves a confirmation that the task corresponding to the user's requests has been accomplished to the client (happens when the user says *yes* to a confirmation question).
- The Scheduler retrieves a conflict declaration to the client. Even though the task has not been accomplished from the dialogue manager point of view, it has been from the Scheduler's side.

An episode is a portion of a dialogue that starts with an open question (where the user is supposed to utter a complete request with all the necessary slot values) and ends with either a new open question or a user hang up.

7.1.3 Fitted-Q Value Iteration

Fitted-Q iteration has already been successfully applied to dialogue management in the traditional sense [7]. Here it is applied to the problem of turn-taking.

where $\theta(a)$ is a parameter vector associated with action a. For the notations to be more compact, we will omit the action and refer to this parameters vector as θ. The aim of Fitted-Q algorithm is to estimate those parameters for the optimal Q-function Q^*. Recall that the Bellman optimality equation states that

$$Q^*(s,a) = \mathbb{E}[R(s,a,s') + \gamma \max_{a'} Q^*(s,a')|s,a]$$

$$Q^* = T^*Q^*$$

The operator T^* is a contraction [5]. As a consequence, there is a way to estimate it in an iterative way: *Value Iteration* (Banach theorem). Each new iteration is linked to the previous one as follows:

$$\hat{Q}_i^* = T^*\hat{Q}_{i-1}^*$$

However, an exact representation of the Q-function is assumed which is not possible in our case as the state space is infinite (even if we discretise the ASR score, it will still be too big). As a consequence, we adopt a linear representation of the Q-function:

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

$\hat{Q}$ is the projection of Q on the space of the functions that can be written as a linear combination of the state vector's components. Let Π be the corresponding projection operator, then it can be shown that ΠT^* is still a contraction and admits a unique fixed point that can be iteratively computed as follows: $\hat{Q}_{\theta_i} = \Pi T^* \hat{Q}_{\theta_{i-1}}$. As the transition probabilities of the MDP and the reward function are not known, a sampled operator $\hat{T}$ is used instead of T. For a transition (s,a,r,s'), it is defined as

$$\hat{T}\hat{Q}(s,a) = r + \gamma \max_{a}^{\prime} Q(s', a')$$

The Fitted-Q algorithm therefore estimates the θ vector using the iteration rule: $\hat{Q}_{\theta_i} = \hat{T}\hat{Q}_{\theta_{i-1}}$. To compute the projection, the L_2 norm is minimised:

$$\theta_i = \underset{\theta \in \mathbb{R}^p}{\arg\min} \sum_{j=1}^{N} (r_j + \gamma \max_{a} \theta_{i-1}(a)^T \phi s_j - \theta(a_j)\phi(s_j))$$

where N is the number of transitions in the data batch. This is a classic least square optimisation and θ_i can be computed as follows (the matrix inversion has to be performed only once and not at every iteration):

$$\theta_i = (\sum_{i=1}^{N} \phi(s_i)^T \phi(s_i))^{-1} \sum_{j=1}^{N} \phi(s_j)(r_j + \gamma \max_{a} \theta_{i-1}(a)^T \phi(s_j))$$

7.2 Experiment

7.2.1 Setup

The same dialogue scenarios described in Chap. 6 are used here. For learning, the noise level is fixed at 0.15. 50 learning curves have been produced with 3000 episodes each and the average curve is depicted in Fig. 7.1. The θ vectors in the Q-function model are initially zeros and they are updated every 500 episodes. There are three phases to distinguish:

1. **Pure exploration (Episodes 0-500):** The actions are taken randomly with a probability of 0.8 for choosing WAIT and 0.2 for SPEAK. Picking equiprobable actions

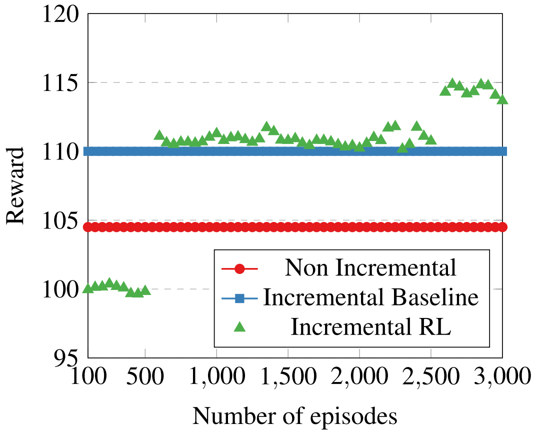


Fig. 7.1 Learning curve (0-500: pure exploration, 500-2500: exploration/exploitation, 2500-3000: pure exploitation)

results in the user being interrupted so often that the interesting scenarios are very rarely explored.

2. **Exploitation/exploration (Episodes 500-2500):** An ε-greedy policy is used with respect to the current Q-function, with $\varepsilon = 0.1$.
3. **Pure exploitation (Episodes 2500-3000):** A 100% greedy policy is used.

7.2.2 Results

Three different strategies are compared:

- **Non-incremental baseline:** It corresponds to the MixIni strategy defined in Chap. 6. The user is asked to provide all the information necessary to execute her request and when there are still missing slots, the corresponding values are asked for one after another.
- **Incremental baseline:** MixIni+Incr from Chap. 6 is selected as an incremental baseline. It is identical to the non-incremental baseline with the difference that it is enhanced with handcrafted turn-taking rules defined in Chap. 6.
- **Incremental RL:** It corresponds to the strategy learned with reinforcement learning.

Like in Chap. [**?**], these strategies are compared under different levels of noise. The non-incremental and the incremental baselines have already been compared in Chap. 6. In Fig. 7.2, they are also compared to the new automatically learned strategy. The differences become clearer as the WER increases. For WER=0.3, the non-incremental baseline reaches 3 minutes, the incremental baseline goes 10 seconds faster and the incremental RL still improves it by an additional 20 seconds (17% gain in total). In terms of task completion, the non-incremental baseline drops under 70%, the incremental baseline shows a performance of 73% whereas the incremental RL keeps the ratio at a level of 80%.

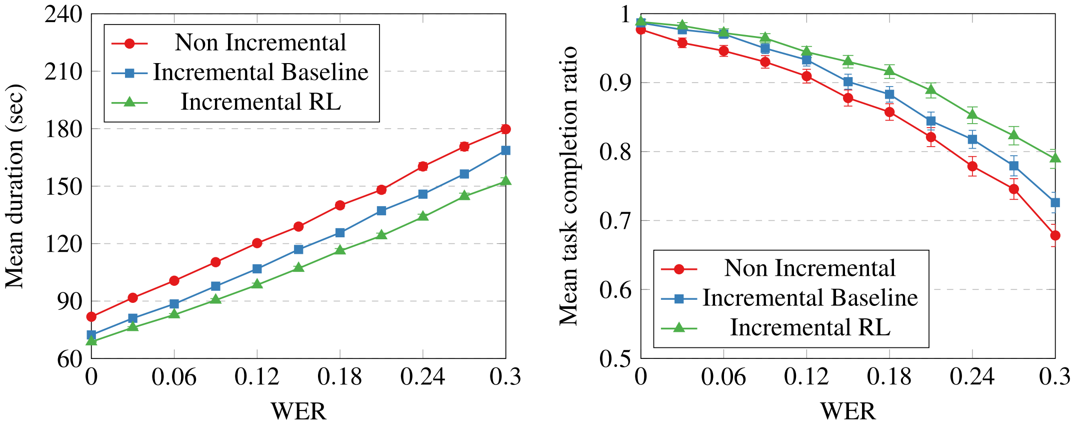


Fig. 7.2 Mean dialogue duration and task for the non-incremental, the baseline incremental and the RL incremental strategies under different noise conditions.

Chapter 8

Experiment with real users

8.1 Protocol

8.2 Results

Chapter 9

A new reinforcement learning approach adapted to incremental dialogue

9.1 Problem

9.2 Proposed approach

9.3 Toy problem application

9.4 Simulation results

Chapter 10

Conclusion

Conclusion here

References

[1] Austin, J. (1962). *How to Do Things with Words*. Oxford.

[2] Baumann, T. and Schlangen, D. (2011). Evaluation and optimisation of incremental processors. *Dialogue and Discourse*, 2:113–141.

[3] Baumann, T. and Schlangen, D. (2013). Open-ended, extensible system utterances are preferred, even if they require filled pauses. In *Proceedings of the SIGDIAL 2013 Conference*.

[4] Beattie, G. (1982). Turn-taking and interruption in political interviews: Margaret thatcher and jim callaghan compared and contrasted. *Semiotica*, 39:93–114.

[5] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

[6] Chandramohan, S., Geist, M., Lefèvre, F., and Pietquin, O. (2011). User simulation in dialogue systems using inverse reinforcement learning. In *INTERSPEECH*.

[7] Chandramohan, S., Geist, M., and Pietquin, O. (2010). Optimizing spoken dialogue management with fitted value iteration. In *INTERSPEECH 11th Annual Conference of the International Speech*.

[8] Dethlefs, N., Hastie, H. W., Rieser, V., and Lemon, O. (2012). Optimising incremental dialogue decisions using information density for interactive systems. In *EMNLP-CoNLL*.

[9] Eckert, W., Levin, E., and Pieraccini, R. (1997). User modeling for spoken dialogue system evaluation. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*.

[10] El Asri, L., Lemonnier, R., Laroche, R., Pietquin, O., and Khouzaimi, H. (2014). NASTIA: Negotiating Appointment Setting Interface. In *Proceedings of LREC*.

[11] Gasic, M., Breslin, C., Henderson, M., Kim, D., Szummer, M., Thomson, B., Tsiakoulis, P., and Young, S. (2013). Pomdp-based dialogue manager adaptation to extended domains. In *Proceedings of the SIGDIAL 2013 Conference*.

[12] Ghigi, F., Eskenazi, M., Torres, M. I., and Lee, S. (2014). Incremental dialog processing in a task-oriented dialog. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[13] Gravano, A. and Hirschberg, J. (2011). Turn-taking cues in task-oriented dialogue. *Comput. Speech Lang.*, 25(3):601–634.

[14] Hastie, H., Aufaure, M.-A., Alexopoulos, P., Cuayáhuitl, H., Dethlefs, N., Gasic, M., Henderson, J., Lemon, O., Liu, X., Mika, P., Ben Mustapha, N., Rieser, V., Thomson, B., Tsiakoulis, P., and Vanrompay, Y. (2013). Demonstration of the parlance system: a data-driven incremental, spoken dialogue system for interactive search. In *Proceedings of the SIGDIAL 2013 Conference*.

[15] Jonsdottir, G. R., Thorisson, K. R., and Nivel, E. (2008). Learning smooth, human-like turntaking in realtime dialogue. In *In Proceedings of Intelligent Virtual Agents (IVA 08*, pages 162–175. Springer.

[16] Khouzaimi, H., Laroche, R., and Lefèvre, F. (2014). An easy method to make dialogue systems incremental. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

[17] Kim, S. and Banchs, R. E. (2014). Sequential labeling for tracking dynamic dialog states. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

[18] Laroche, R. (2010). *Raisonnement sur les incertitudes et apprentissage pour les systemes de dialogue conventionnels*. PhD thesis, Paris VI University.

[19] Laroche, R., Putois, G., Bretier, P., Aranguren, M., Velkovska, J., Hastie, H., Keizer, S., Yu, K., Jurcicek, F., Lemon, O., and Young, S. (2011). D6.4: Final evaluation of classic towninfo and appointment scheduling systems. Report D6.4, CLASSIC Project.

[20] Lemon, O., Cavedon, L., and Kelly, B. (2003). Managing dialogue interaction: A multi-layered approach. In *Proceedings of the 4th SIGDIAL Workshop on Discourse and Dialogue*.

[21] Lemon, O. and Pietquin, O. (2007). Machine learning for spoken dialogue systems. In *Proceedings of the European Conference on Speech Communication and Technologies (Interspeech'07)*.

[22] Levin, E., Pieraccini, R., and Eckert, W. (1997). Learning dialogue strategies within the markov decision process framework. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*.

[23] Lock, K. (1965). Structuring programs for multiprogram time-sharing on-line applications. In *AFIPS '65 (Fall, part I) Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*.

[24] McGraw, I. and Gruenstein, A. (2012). Estimating word-stability during incremental speech recognition. In *Proceedings of the INTERSPEECH 2012 Conference*.

[25] Meena, R., Skantze, G., and Gustafson, J. (2013). A data-driven model for timing feedback in a map task dialogue system. In *Proceedings of the SIGDIAL 2013 Conference*.

[26] Pietquin, O. and Hastie, H. (2013). A survey on metrics for the evaluation of user simulations. *The Knowledge Engineering Review*, 28.

[27] Plátek, O. and Jurčíček, F. (2014). Free on-line speech recogniser based on kaldi asr toolkit producing word posterior lattices. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

[28] Raux, A. and Eskenazi, M. (2007). A multi-layer architecture for semi-synchronous event-driven dialogue management. In *ASRU*.

[29] Raux, A. and Eskenazi, M. (2008). Optimizing endpointing thresholds using dialogue features in a spoken dialogue system. In *SIGDIAL*.

[30] Sacks, H., Schegloff, E. A., and Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, 50:696–735.

[31] Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. (2007). Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*.

[32] Schatzmann, J., Weilhammer, K., Stuttle, M., and Young, S. (2006). A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The Knowledge Engineering Review*, 21.

[33] Schlangen, D. and Skantze, G. (2011). A general, abstract model of incremental dialogue processing. *Dialogue and Discourse*, 2:83–111.

[34] Searle, J. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, UK.

[35] Selfridge, E. and Heeman, P. A. (2010). Importance-driven turn-bidding for spoken dialogue systems. In *ACL*, pages 177–185.

[36] Selfridge, E. O., Arizmendi, I., Heeman, P. A., and Williams, J. D. (2011). Stability and accuracy in incremental speech recognition. In *Proceedings of the SIGDIAL 2011 Conference*.

[37] Selfridge, E. O., Arizmendi, I., Heeman, P. A., and Williams, J. D. (2012). Integrating incremental speech recognition and pomdp-based dialogue systems. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*.

[38] Selfridge, E. O. and Heeman, P. A. (2012). A temporal simulator for developing turn-taking methods for spoken dialogue systems. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*.

[39] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning, An Introduction*. The MIT Press, Cambridge, Massachusetts, London, England.

[40] Tanenhaus, M. K., Spivey-Knowlton, M. J., Eberhard, K. M., and Sedivy, J. C. (1995). Integration of visual and linguistic information in spoken language comprehension. *Science*, 268:1632–1634.

[41] Williams, J. D. (2012). A belief tracking challenge task for spoken dialog systems. In *NAACL HLT 2012 Workshop on Future directions and needs in the Spoken Dialog Community: Tools and Data*.

[42] Williams, J. D. and Young, S. (2007). Partially observable markov decision processes for spoken dialog systems. *Comput. Speech Lang.*

[43] Wirén, M. (1992). *Studies in Incremental Natural Language Analysis*. PhD thesis, Linköping University, Linköping, Sweden.

[44] Wlodarczak, M. and Wagner, P. (2013). Effects of talk-spurt silence boundary thresholds on distribution of gaps and overlaps. In *INTERSPEECH Proceedings*.

[45] Yuan, J., Liberman, M., and Cieri, C. (2006). Towards an integrated understanding of speaking rate in conversation. In *INTERSPEECH Proceedings*.

Appendix A

How to install LaTeX

Windows OS

TeXLive package - full version

1. Download the TeXLive ISO (2.2GB) from https://www.tug.org/texlive/
2. Download WinCDEmu (if you don't have a virtual drive) from http://wincdemu.sysprogs.org/download/
3. To install Windows CD Emulator follow the instructions at http://wincdemu.sysprogs.org/tutorials/install/
4. Right click the iso and mount it using the WinCDEmu as shown in http://wincdemu.sysprogs.org/tutorials/mount/
5. Open your virtual drive and run setup.pl

or

Basic MikTeX - TeX distribution

1. Download Basic-MiKTeX(32bit or 64bit) from http://miktex.org/download
2. Run the installer
3. To add a new package go to Start » All Programs » MikTex » Maintenance (Admin) and choose Package Manager

4. Select or search for packages to install

TexStudio - TeX editor

1. Download TexStudio from http://texstudio.sourceforge.net/#downloads
2. Run the installer

Mac OS X

MacTeX - TeX distribution

1. Download the file from https://www.tug.org/mactex/
2. Extract and double click to run the installer. It does the entire configuration, sit back and relax.

TexStudio - TeX editor

1. Download TexStudio from http://texstudio.sourceforge.net/#downloads
2. Extract and Start

Unix/Linux

TeXLive - TeX distribution

Getting the distribution:

1. TexLive can be downloaded from http://www.tug.org/texlive/acquire-netinstall.html.
2. TexLive is provided by most operating system you can use (rpm,apt-get or yum) to get TexLive distributions

Installation

1. Mount the ISO file in the mnt directory

   ```
   mount -t iso9660 -o ro,loop,noauto /your/texlive####.iso /mnt
   ```

2. Install wget on your OS (use rpm, apt-get or yum install)
3. Run the installer script install-tl.

   ```
   cd /your/download/directory
   ./install-tl
   ```

4. Enter command ‘i’ for installation
5. Post-Installation configuration:
   http://www.tug.org/texlive/doc/texlive-en/texlive-en.html#x1-320003.4.1
6. Set the path for the directory of TexLive binaries in your .bashrc file

For 32bit OS

For Bourne-compatible shells such as bash, and using Intel x86 GNU/Linux and a default directory setup as an example, the file to edit might be

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/i386-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

For 64bit OS

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/x86_64-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
```

```
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

Fedora/RedHat/CentOS:

```
sudo yum install texlive
sudo yum install psutils
```

SUSE:

```
sudo zypper install texlive
```

Debian/Ubuntu:

```
sudo apt-get install texlive texlive-latex-extra
sudo apt-get install psutils
```

Appendix B

Installing the CUED class file

LaTeX.cls files can be accessed system-wide when they are placed in the <texmf>/tex/latex directory, where <texmf> is the root directory of the user's TeXinstallation. On systems that have a local texmf tree (<texmflocal>), which may be named “texmf-local” or “localtexmf”, it may be advisable to install packages in <texmflocal>, rather than <texmf> as the contents of the former, unlike that of the latter, are preserved after the LaTeXsystem is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory <texmf>/tex/latex/CUED for all CUED related LaTeXclass and package files. On some LaTeXsystems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For TeXLive systems this is accomplished via executing “texhash” as root. MIKTeXusers can run “initexmf -u” to accomplish the same thing.

Users not willing or able to install the files system-wide can install them in their personal directories, but will then have to provide the path (full or relative) in addition to the filename when referring to them in LaTeX.