

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/257267620>

Towards incremental speech generation in conversational systems

Article in *Computer Speech & Language* · January 2013

DOI: 10.1016/j.csl.2012.05.004

CITATIONS

49

READS

437

2 authors:



Gabriel Skantze

KTH Royal Institute of Technology

182 PUBLICATIONS 3,231 CITATIONS

[SEE PROFILE](#)



Anna Hjalmarsson

KTH Royal Institute of Technology

38 PUBLICATIONS 936 CITATIONS

[SEE PROFILE](#)

Towards incremental speech generation in conversational systems[☆]

Gabriel Skantze*, Anna Hjalmarsson

Department of Speech Music and Hearing, KTH, Sweden

Received 15 December 2011; received in revised form 16 May 2012; accepted 19 May 2012

Available online 30 May 2012

Abstract

This paper presents a model of incremental speech generation in practical conversational systems. The model allows a conversational system to incrementally interpret spoken input, while simultaneously planning, realising and self-monitoring the system response. If these processes are time consuming and result in a response delay, the system can automatically produce hesitations to retain the floor. While speaking, the system utilises hidden and overt self-corrections to accommodate revisions in the system. The model has been implemented in a general dialogue system framework. Using this framework, we have implemented a conversational game application. A Wizard-of-Oz experiment is presented, where the automatic speech recognizer is replaced by a Wizard who transcribes the spoken input. In this setting, the incremental model allows the system to start speaking while the user's utterance is being transcribed. In comparison to a non-incremental version of the same system, the incremental version has a shorter response time and is perceived as more efficient by the users.

© 2012 Elsevier Ltd. All rights reserved.

Keywords: Conversational systems; Incremental processing; Speech generation; Wizard-of-Oz

1. Introduction

Speakers in dialogue produce speech in a piece-meal fashion as the dialogue progresses. When starting to speak, dialogue participants typically do not have a complete plan of how to say something or even what to say. Yet, they manage to rapidly integrate information from different sources in parallel and simultaneously plan and realise new dialogue contributions (Levelt, 1989). Processes at all levels (e.g. semantic, syntactic, phonologic and articulatory) work in parallel to interpret incoming speech from interlocutors and at the same time render new responses. Kempen and Hoenkamp (1982, 1987) refer to this phenomenon as *incremental processing*.

Contrary to this, conversational systems of today typically process the dialogue one utterance at a time, and each module of the system has to complete the processing of the utterance before passing the result on to the next module. This is illustrated in the left pane in Fig. 1. Such non-incremental processing is associated with several problems. For instance, non-incremental systems cannot utilise any higher level information for detecting relevant places to take the turn. Instead, they often rely on silence detection and a time-out. Silence, however, is a crude indicator of ends of turns. For example, if a relatively short silence threshold is used, the system risks interrupting users that pause momentarily

[☆] This paper has been recommended for acceptance by Bernd Moebius.

* Corresponding author. Tel.: +46 87907874.

E-mail addresses: gabriel@speech.kth.se (G. Skantze), annah@speech.kth.se (A. Hjalmarsson).

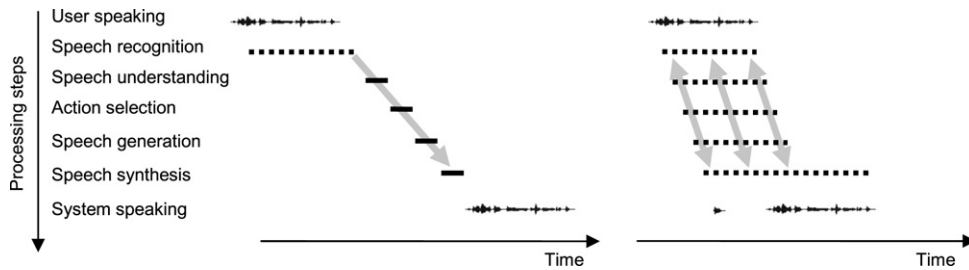


Fig. 1. Schematic view of the difference between non-incremental dialogue processing (left pane) and incremental processing (right pane). Dashed lines indicate incremental processing. As the left pane shows, speech recognition is typically done incrementally as the user is speaking, but the intermediate results are not used.

in the middle of a turn. On the other hand, if the system uses a long silence threshold, there will be a long latency before the system responds.

In an incremental system, processing starts before the input is complete and the first output increments are produced as soon as possible (Kilger and Finkler, 1995). This means that an incremental conversational system should not wait until the user has finished speaking before processing starts, but that processing should start already when smaller units (such as words) are identified in the input. All modules of the system then asynchronously process increments on different levels of processing. This is illustrated in the right pane in Fig. 1. As the figure shows, incremental processing allows the system to produce feedback as the user is speaking, to utilise higher-level information for deciding when to take the turn, and to let higher-level information (such as semantics) guide lower-level-processing (such as speech recognition). While incremental processing allows for a more sophisticated interaction model, it also poses new challenges. The incremental output of a system module will often be tentative, which means that the module might have to revise the output later on, possibly causing a chain of revisions in the whole system and possibly in the system's spoken output (Schlangen and Skantze, 2011).

There have been several attempts to improve dialogue systems or components by using incremental processing. DeVault et al. (2009) presented a study where a machine learning approach was used to detect at what point in the user's utterance the system could "guess" how it would end, thus allowing the system to interrupt the user and complete the utterance. In Heintze et al. (2010), statistical methods for incremental natural language understanding were explored. A completely incremental dialogue system was presented by Skantze and Schlangen (2009). The system was limited to number dictation, but unlike more traditional dictation systems, where the system does not respond until the complete number sequence has been read, the system used incremental speech recognition and prosodic analysis to give rapid feedback as the user was reading the number sequence. A user study showed that the incremental system was preferred over a non-incremental version of the same system.

The benefits of incremental processing might be more easily identified on the input side of conversational systems. In this paper, however, we will also show how incremental processing on the output side can be beneficial. We will show how incremental speech generation can be used to improve the response time of the system and improve the user experience. The paper is structured as follows. First, we will put forth some arguments for incremental speech generation, discuss related work and identify some requirements for conversational systems. Second, we will present a model of incremental speech generation that allows the system to incrementally interpret spoken input, while simultaneously planning, realising and monitoring its own speech production. Third, we will present a user study in a Wizard-of-Oz setting, where the model is implemented in a practical conversational system.

2. Incremental speech generation

2.1. Motivation

Allen et al. (2001) discuss some important shortcomings of non-incremental processing when modelling conversational human-like dialogue. One observation is that people often seem to start to speak before knowing exactly what to say next (possibly to grab the turn), thus planning and producing the utterance incrementally. As illustrated in Fig. 1,

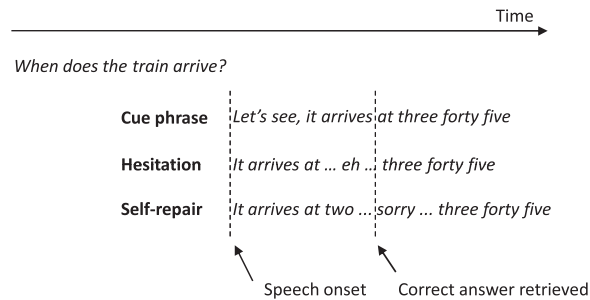


Fig. 2. Three different ways of coping with delayed processing while generating speech.

an incremental system can, in a similar manner, start to plan the response while the user is still speaking. When the system detects a relevant place to take the turn, it starts to asynchronously realise this plan while processing continues, with the possibility to revise the plan if needed.

This concurrent planning and realisation of the spoken output can be used to respond more quickly. For example, if some external resource such as a database needs to be accessed, the system can produce a tentative, incomplete plan and start to speak before the complete response has been generated. Fig. 2 shows three examples of how this can be done. In the first example, the system starts to say “let’s see, it arrives” before the answer is retrieved, since this segment can be generated without knowing the exact time. The second example shows how a filled pause (“eh”) can be inserted to maintain the floor, when the first part of a speech plan has been realised and the database result needed to plan the rest of the utterance is delayed for some reason. In the third example, the system makes a “guess” (which might for example be based on a cached result from a previous dialogue), which then might need to be repaired if the actual response differs from the one produced.

As the example above illustrates, incremental speech generation opens up new computational possibilities since the system can be allowed additional time for processing. For example, it has been assumed that ASR processing has to be done in real-time, in order to avoid long and confusing response delays. Yet, if we allow the system to start speaking before input is complete, we can allow more accurate (and time-consuming) ASR processing, for example by broadening the beam.

Another situation where processing may take time, which we will explore in this paper, is in a Wizard-of-Oz setting, where a human operator simulates parts of the system. A common problem in such settings is the time it takes for the Wizard to interpret the user’s utterance and/or decide on the next system action, resulting in unacceptable response delays (Fraser and Gilbert, 1991). Thus, it would be useful if the system could start to speak as soon as the user has finished speaking, based on the Wizard’s actions so far.

Another aspect of incremental speech generation, identified by Allen et al. (2001), is that people often give feedback in the middle of utterances in the form of acknowledgements, repetition of fragments or clarification requests. Furthermore, when a speaker is interrupted or receives feedback in the middle of an utterance, he is able to continue the utterance from the point where he was interrupted. To be able to model this behaviour, the system must incrementally monitor the system’s own as well as the user’s speech production. If the user interrupts the system, the dialogue manager needs to know what parts of the intended output were actually realised. Also, the timing of the synthesized speech is important, in order to understand feedback from the user in the middle of system utterances.

2.2. Related work

The incremental approach to speech processing outlined above is inspired by psycholinguistics and theories of human speech processing. Conversational speech is spontaneous and produced in real-time. The resources for planning of future responses are limited by time restrictions and humans as well as machines need to deal with the time constraints of this setting. Empirical studies have used reaction times of different language production tasks to study what type of units the different levels of processing operate on and to what extent speakers plan ahead. It has been shown that hesitation phenomena are more likely to occur before longer utterances (Shriberg, 1994). These findings and spoonerisms such as “cuff of coffee” (Fromkin, 1973), where parts of an utterance or a word has been switched around, suggest that

planning (at least) goes beyond the immediate word. Still, while speakers may not produce speech phone by phone or word by word, there are findings which show that planning affects articulation, suggesting that these processes occur almost simultaneously. For example, in a sentence production task that included arithmetic calculations, the duration of utterances was affected by problem difficulty (Ferreira and Swets, 2002). Moreover, Brysbaert and Fias (1998) showed that the response time for answers to arithmetical problems was shorter when the problem was presented so that the first term in the calculation corresponded to the first phonological word uttered. Whereas the intermediate levels of speech production and the units of processing are still under debate (c.f. Caramazza, 1997), the literature referenced above provides empirical evidence for the incrementality of speech production.

The fact that incremental speech processing occurs on several different levels simultaneously allows speakers to react instantly to events that change the current dialogue context. Thus, while talking, speakers may obtain new information that forces them to revise and refine their utterance plan as they go along. This ongoing supervision is referred to as *speech monitoring* (for an overview see Postma, 2000). The speech monitoring system needs to supervise both internal and external events in order to be able to adjust to errors in the speaker's own articulation as well as to clarification requests and interruptions from the interlocutor. Monitoring of the speaker's own production – so-called *self-monitoring* – appears to be done both *covertly* (before articulation) and *overtly* (after articulation) (c.f. Levelt, 1989). Since covert repairs are done before articulation, the utterance plan can be altered without the listeners noticing. However, for overt repairs, the speaker needs to indicate how the previous segment was altered. This can be done by marking the repair prosodically (c.f. Howell and Young, 1991) or using an editing term, such as “sorry” or “I mean”. Furthermore, Blackmer and Mitton (1991) show that 12.4% of the cut-off-to-repair times – the duration of time between a cut-off to the actual repair – were zero milliseconds. Since the error segment and its repair are likely not conceptualized as a single unit, self-monitoring appears to occur simultaneously as articulation. This suggests that the processes of self-monitoring also operate in an incremental fashion.

Kempen and Hoenkamp (1987) present one of the first attempts to model incremental sentence formulation in humans computationally. They propose Incremental Procedural Grammar as a suitable grammar formalism. Another approach, using Tree Adjoining Grammar, is presented in Kilger and Finkler (1995). The authors also discuss how to incorporate overt and covert (or hidden) repairs in their model. The focus of these studies is on syntactic surface realisation. A computational model for deep generation on the conceptual level is presented in Guhe (2007).

The focus of the model presented in this paper will not be on conceptual or syntactic generation as isolated tasks, but on how to integrate incremental speech generation with the other components in a complete incremental conversational system, in order to cope with the events of the dialogue.

An architecture for incremental speech generation in conversational systems is presented in Dohsaka and Shimazu (1997). Their focus is how to incrementally produce utterances and simultaneously incorporate input from users that occurs in the middle of the system's speech. Similarly, in the incremental number dictation system presented by Skantze and Schlangen (2009), the system may listen for user reactions (acknowledgements or corrections) as the system is reading back a number sequence. Thus, the system performs some sort of self- and other-monitoring while speaking.

While these systems can cope with input from the user as the system is speaking, none of them deal with the need for revision in the output as the system is speaking.

2.3. Requirements for conversational systems

The discussion above shows that there are a number of issues that need to be taken into consideration in a model of incremental speech generation, as compared to a traditional, non-incremental approach.

2.3.1. Generation and realisation

In a non-incremental system, the system first decides to say something and formulates a complete response, which is then synthesized and played back to the user. After the utterance is completely realised, the system can make another decision and realise another utterance. In an incremental model, the system should be able to concurrently plan and realise the spoken output. It should not have to generate the complete response before starting to speak. The system should be able to produce hesitations and “floor-holders” if it does not know exactly what to say.

2.3.2. *Units of processing*

The smallest unit of the spoken output in a non-incremental system is the complete utterance, which corresponds to some semantic structure or update of the dialogue state. In an incremental model, the utterance should be broken down into smaller units, both on the speech level and on the conceptual level.

2.3.3. *Self-monitoring*

When a non-incremental system has initiated the response, it assumes that the complete response will be realised. If the utterance needs to be aborted, the system does not know what parts of the utterance has been realised. It must either treat the utterance as completed or basically unsaid. An incremental system should monitor the production of the sub-utterance units and be able to semantically model partial completion of the spoken output.

2.3.4. *Other-monitoring*

If the user speaks while a non-incremental system is speaking (e.g., producing a backchannel), the system cannot model which part of the system's utterance this was a response to. An incremental system should be able to monitor user reactions while it is speaking and relate them to the different parts of the system's utterance.

2.3.5. *Interruptions*

If a non-incremental system is interrupted in the middle of the utterance (e.g., by a cough), it has to either abort or repeat the whole utterance. An incremental system should be able to pause in the middle of an utterance when audio from the user is detected, and then possibly continue where it left off.

2.3.6. *Self-repairs*

In a non-incremental system, no new decisions are being made about the output while the utterance is realised. Therefore, there is no situation where (mid-utterance) self-repairs would be an issue. In an incremental system, new events may occur as the system is speaking that changes the current context, and the system may need to adjust its speech plan to accommodate this change. Therefore, the system should be able to produce self-repairs.

2.3.7. *Speech synthesis*

In an incremental system, the speech synthesizer needs to provide information on how the different sub-units of the input relate to the different points in time in the generated audio, in order to facilitate interruptions, self-monitoring and self-repairs.

3. A model of incremental speech generation

The model we present here is mainly intended for building practical conversational systems, and should not be regarded as a computational or cognitive model of human speech production. However, the model is intended to allow conversational systems to adopt human-like behaviour and it is therefore partly inspired by human speech production. The model is not implementation-specific, but a short description of our implementation of the model will be given in [Section 3.6](#).

The exposition will move from abstract to concrete. We will start by briefly describing a general, abstract model of incremental processing which will serve as a basis for our incremental generation model. We will then describe how the responsibilities of the different modules of the system should be shared and how information at different levels of processing might best be broken down into incremental units. Using these building blocks, we will describe how the system incrementally monitors its own speech production and the user's reaction to it. We also show how the system copes with delays and revisions in its own processing by utilising self-repairs.

3.1. *The IU-model of incremental processing*

In [Schlangen and Skantze \(2011\)](#), a general, abstract model of incremental dialogue processing is presented. In this model, a system consists of a network of processing modules. Each module has a left buffer, a processor, and a right buffer. The normal mode of processing is to receive input from the left buffer, process it, and provide output in the right buffer, from where it is forwarded to the next module's left buffer. An abstract example is shown in [Fig. 3](#). Modules

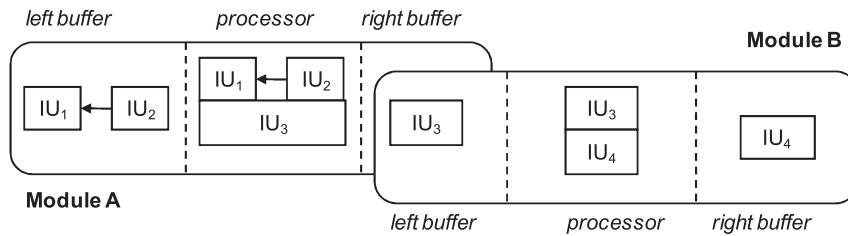


Fig. 3. Two connected modules.

exchange *incremental units* (IUs), which are the smallest chunks of information that can trigger connected modules into action (such as words, phrases, communicative acts). IUs are typically part of larger units: individual words are parts of an utterance; concepts are part of the representation of an utterance meaning. This relation of being part of the same larger unit is recorded through *same-level links*. In the example in Fig. 3, IU₂ has a same-level link to IU₁ of type PREDECESSOR, meaning that they are linearly ordered. The information that was used in creating a given IU is linked to it via *grounded-in links*. In the example, IU₃ is grounded in IU₁ and IU₂, while IU₄ is grounded in IU₃.

An incremental processing module may have to produce a tentative result before “seeing the whole picture”, and then later discover that it was wrong, as more input units are processed. Thus, in an incremental system, there is potentially a need for *revision*. This can be exemplified with an automatic speech recognizer (ASR). In a non-incremental ASR, a chunk of audio is first identified as an utterance (often using simple silence detection), and the ASR then finds an optimal sequence of words that best matches the audio chunk. In an incremental ASR, as shown in Fig. 4, the currently best hypothesis may change for each new audio frame that is consumed. When the first audio frames of the word ‘forty’ are consumed, the best hypothesis might be ‘four’. As the ASR consumes more audio frames, it might need to revise its previous output. If other modules have started to process the tentative hypotheses, this might cause a chain of revisions in all subsequent modules.

To handle revision, modules have to be able to react to three basic situations. First, IUs may be **added** to the left buffer, which triggers processing and possibly results in a new IU being posted on the right buffer. Second, IUs that were erroneously hypothesized by an earlier module may be **revoked**, which may trigger a revision of a module’s own output. By following the grounded-in links, a module may discover that an IU no longer has any support (i.e., it has been revoked). Finally, modules must at some point **commit** to IUs, that is, ensure that they will not be revoked. This is what happens at t_3 in Fig. 4. Commitment allows other modules to free up resources, and to commit to their output.

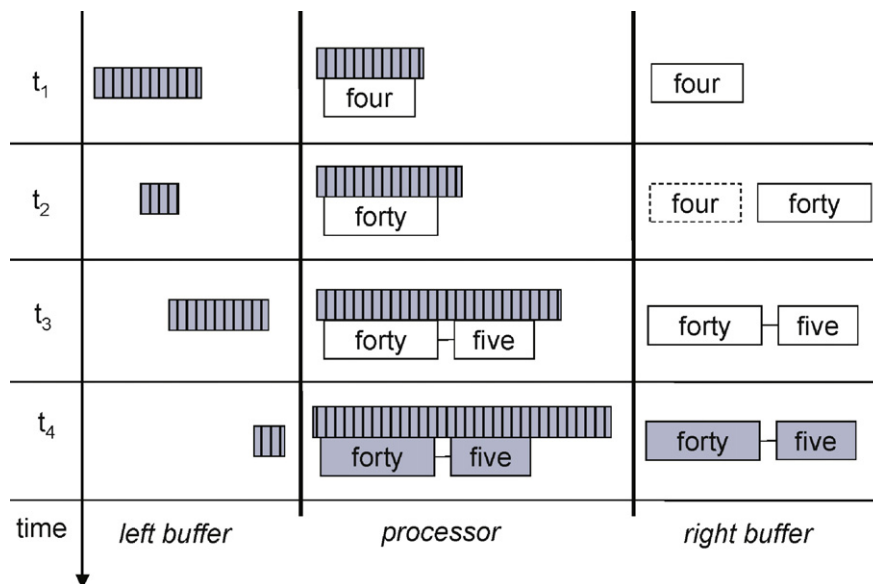


Fig. 4. Speech recognition as an example of incremental processing. Dotted outlines represent revoked units and shaded boxes represent commitment.

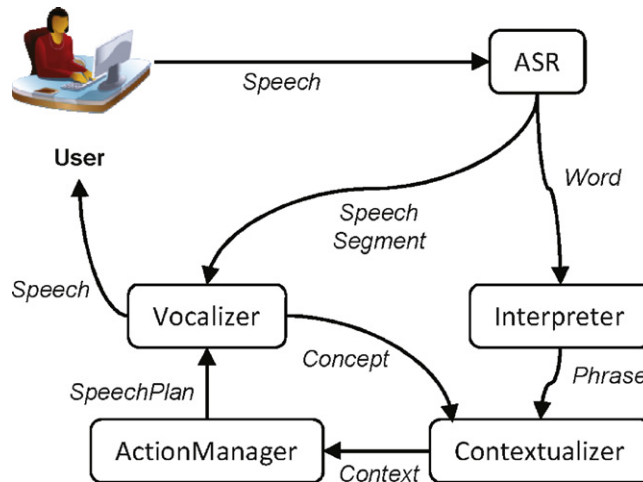


Fig. 5. The modularization for the proposed model.

The model outlined above is general in the sense that non-incremental systems can be regarded as a special case of the model, where the smallest unit of processing is the utterance and where no revision takes place.

3.2. Modularization

When designing the architecture for a conversational system, it is very important to consider how the responsibilities of the different modules should be divided. We will here propose a modularisation that is similar to a typical dialogue system, with some exceptions. It is roughly based on the Higgins architecture described in Skantze (2007). The proposed architecture is shown in Fig. 5. The boxes represent modules and the arrows buffers between the modules.

- The **Recognizer** takes speech as input and produces words that are forwarded to the interpreter. It also detects silence in the speech signal in order to find possible places to give responses. This information is sent directly to the Vocalizer.
- The **Interpreter** parses the words from the Recognizer and finds sequences of syntactic phrases with their corresponding semantic concepts, without taking the dialogue context into account. These phrases are sent to the Contextualizer.
- The **Contextualizer** builds a **Context model**, consisting of **Communicative Acts** (CAs) from the user and the system. Since the Interpreter does not take the dialogue history into account, the Contextualizer uses the current Context model to reinterpret concepts from the Interpreter and add them to the Context model. As can be seen in the figure, there is also a self-monitoring loop between the Vocalizer and the Contextualizer. Thus, as soon as the system has uttered something, the associated concepts are sent to the Contextualizer which adds them to the Context model.
- The **Action Manager** (AM) consults the last user CA in the Context model and makes decisions about the system's future behaviour, such as deciding when and how to make a response. To do this, the AM produces a **Speech Plan**, which is a graph that contains possible surface realisations of the utterance. The Speech Plan is then sent to the Vocalizer.
- The responsibility of the **Vocalizer** is to realise the Speech Plan (i.e., producing speech), taking time-critical events into account. The Vocalizer also incrementally self monitors its own output in order to do this. This self-monitoring facilitates the following things:
 - The Vocalizer incrementally informs the Contextualizer about what the system has said, on the concept level.
 - If speech is detected from the user, the Vocalizer can pause in the middle of the Speech Plan. If the action manager does not decide to respond to the spoken input (it might for example be regarded as a backchannel), the Vocalizer might resume the output from where it left off before the interruption.

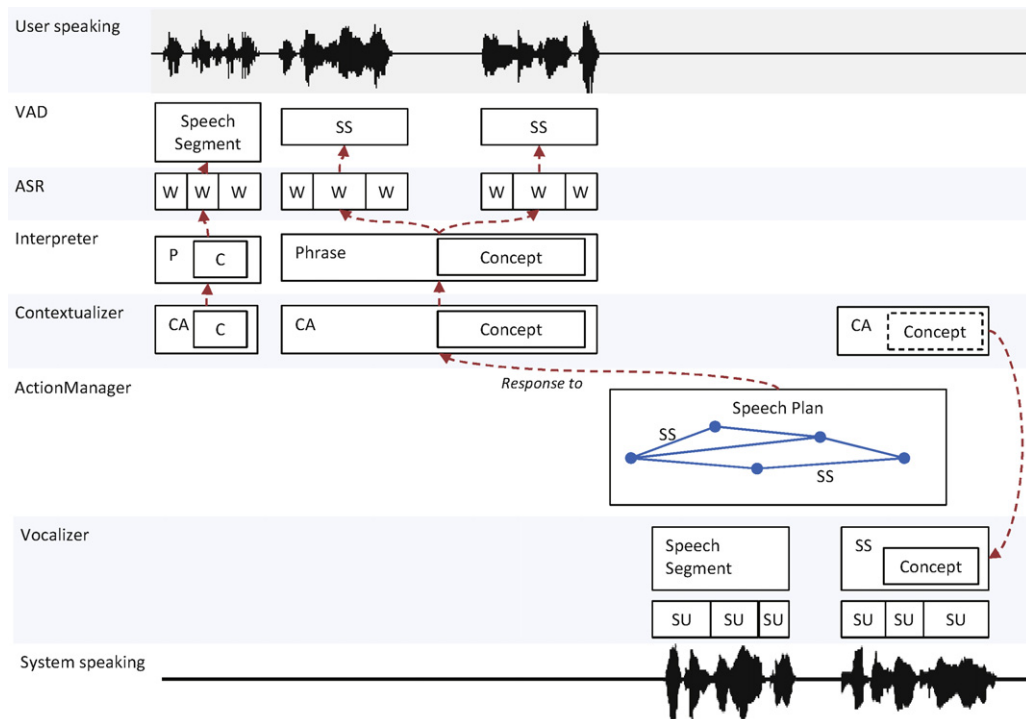


Fig. 6. Incremental Units at different levels of processing. Some grounded-in relations are shown with dotted lines. W = Word, SS = SpeechSegment, SU = SpeechUnit, CA = Communicative Act, P = Phrase. As shown in the figure, the Contextualizer models utterances from both the user and the system on a semantic level. Not all system Speech Segments are associated with concepts (such as fillers).

- If there is a revision in the system and a user CA is revoked, this might result in a revision of the current Speech Plan. The Vocalizer might then need to make mid-utterance self-repairs.

In the proposed architecture, there is no dedicated module for concept-to-text processing, since this is done by the AM. It is of course possible to modularize this task and insert a **Formulator** between the AM and the Vocalizer.

3.3. Incremental units

In order for user and system utterances to be interpreted and produced incrementally, they need to be decomposed into smaller units of processing (IUs). This decomposition is shown in Fig. 6. A voice activity detector (in the Recognizer module) segments the user's speech into **Speech Segments**, using a short silence threshold of 50 ms. This notion of Speech Segments corresponds roughly to the notion of Inter-Pausal Units (Gravano and Hirschberg, 2009). Such a short silence threshold allows the system to give very fast responses (including backchannels), if the AM decides to do so. As can be seen in Fig. 5, information about Speech Segments and their boundaries is sent directly from the Recognizer to the Vocalizer. The Recognizer produces **Word** units, which are also annotated with their start and end times. This way, the grounded-in links can be followed to derive the timing of IUs at different levels of processing. The Interpreter produces syntactic **Phrases** and semantic **Concepts**, which are contextually re-interpreted by the Contextualizer and packaged as Communicative Acts (CAs).

The system output is also modelled using IUs at different processing levels. The widest-spanning IU on the output side is the Speech Plan. The realisation of a Speech Plan will result in a sequence of Speech Segments, where each Speech Segment represents a continuous audio rendering of speech, either as a synthesised string or a pre-recorded audio file. As can be seen in Fig. 6, if the Speech Plan is a response to a user CA, it becomes grounded in it. This allows the system to make revisions in the output when revision occurs in the input, as explained later on.

The Speech Plan is modelled as a directed graph, where each edge is associated with a Speech Segment, as shown in Fig. 7. This allows the Action Manager to asynchronously plan (a set of possible) responses, while the Vocalizer selects

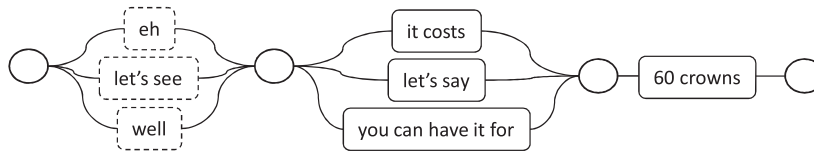


Fig. 7. An example Speech Plan consisting of Speech Segments. Dotted outlines represent optional segments.

the realisation path in the graph and takes care of time-critical synchronization and variation of the system output. To allow the Vocalizer to interrupt and make self-repairs in the middle of a Speech Segment, each Speech Segment can also be decomposed into an array of **Speech Units**, where each Speech Unit contains pointers in time to the audio rendering in the Speech Segment.

When the Action Manager detects a new user CA in the Context model, it may produce a new Speech Plan. As soon as the Vocalizer detects that it is suitable for the system to start speaking, it starts to realise the Speech Plan by traversing the graph. The Vocalizer keeps track of which Speech Segments it has realised before, so that it can look ahead in the graph and avoid recently realised segments, which results in a more varied output.

To control the realisation of the Speech Plan, the Action Manager annotates each Speech Segment with a set of properties:

- **OTHERDELAY**: An integer value that tells the Vocalizer how many milliseconds the system should wait after the last *user* Speech Segment, before realising this Speech Segment. By setting the **OTHERDELAY** of a Speech Segment, the Action Manager can delay the response depending on how certain it is that it is appropriate to speak, for example by considering pitch and semantic completeness in the user's utterance (see [Raux and Eskenazi, 2008](#) for a study on how such dynamic delays can be derived using machine learning).
- **SELFDelay**: An integer value that tells the Vocalizer how many milliseconds the system should wait after the last *system* Speech Segment, before realising this Speech Segment.
- **FINAL**: A boolean value that tells the Vocalizer that the Speech Plan is completely realised. If there are no **FINAL** segments in the Speech Plan, it has not been completely constructed yet.
- **OPTIONAL**: A boolean value that informs the Vocalizer that this Speech Segment does not have to be realised, that is, it does not have any significant semantic or syntactic impact on the realisation. For example, given the context of a price request (“what does it cost?”), the first Speech Segment in the Speech Plan [“it costs”, “30 crowns”] might be optional, since “30 crowns” would be enough to realise a coherent response. **OPTIONAL** Speech Segments are skipped by the Vocalizer if the Speech Plan contains a **FINAL** Speech Segment.

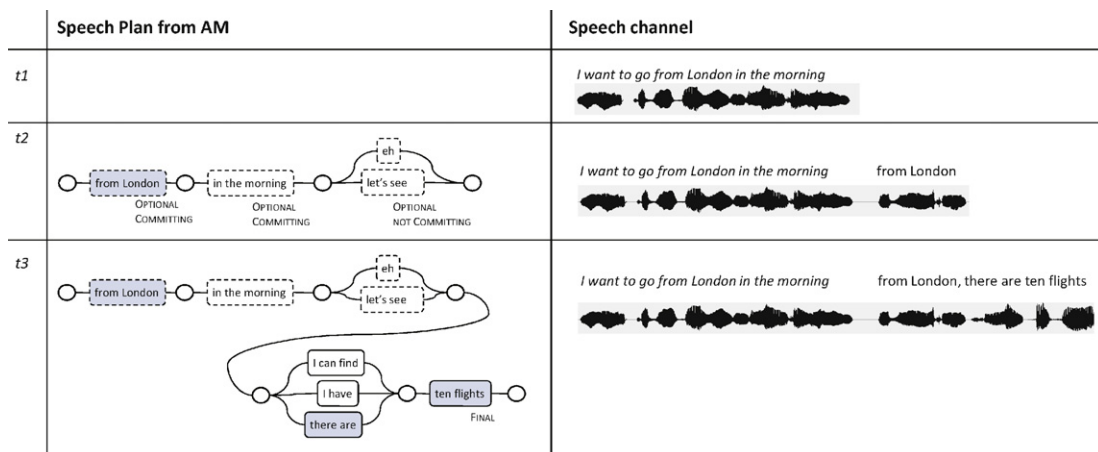


Fig. 8. A Speech Plan is incrementally and concurrently produced and realised.

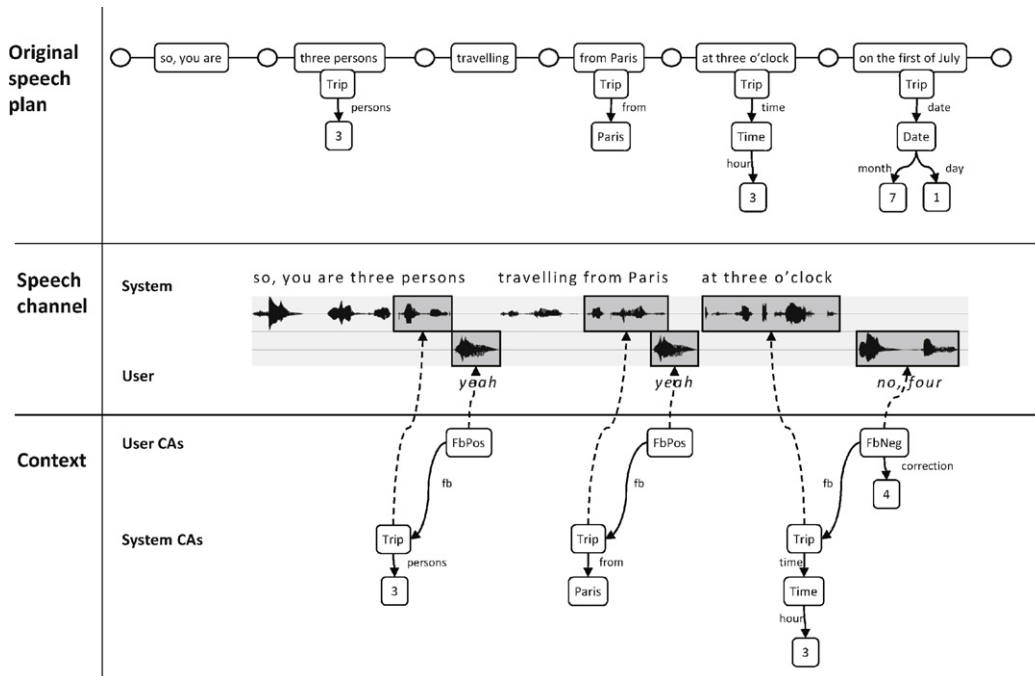


Fig. 9. The Contextualizer's monitoring of the realisation of a Speech Plan. The dotted lines show how the concepts in the context model are grounded in the speech signal.

- **COMMITTING:** A boolean value that informs the Vocalizer that this Speech Segment is committing, that is, needs some kind of editing term (e.g., “sorry”), if the Speech Plan is aborted and repaired, as explained below. Filled pauses, such as “ehm” are typically not committing.
- **CONCEPT:** A semantic concept that is associated with this Speech Segment. This concept is sent from the Vocalizer to the Contextualizer when the Speech Segment has been completely realised.

An example of how a Speech Plan is incrementally produced is shown in Fig. 8. In this example, the system needs to search a database before answering the user's request, and this search takes some time. While querying the database, the AM produces a tentative Speech Plan (at t_2), which contains feedback on the user's request and some floor-holding segments, which are all marked as **OPTIONAL**. Since no **FINAL** segment is in sight, the Vocalizer starts to realise the first **OPTIONAL** segment (“from London”). At t_3 , the AM has retrieved a result from the database, and the Speech Plan is amended. Since the Vocalizer now detects a **FINAL** segment in the Speech Plan, it skips the other **OPTIONAL** segments. The fact that the time of departure (“in the morning”) was never implicitly confirmed with the user is accurately modelled by the Contextualizer, as described in the next section. Note also that the first two optional segments are **COMMITTING**, while the others (“eh” and “let's see”) are not.

3.4. Self- and other-monitoring

The Vocalizer monitors Speech Segments from the user (received directly from Recognizer, as seen in Fig. 5), as well as the Vocalizer's own realisation of Speech Segments. If the user starts to speak as the system is speaking, the Vocalizer pauses (at a Speech Unit boundary) and waits until it has received a new response from the Action Manager. The Action Manager can then choose to generate a new response or simply ignore the last input, in which case the Vocalizer continues from the point of interruption. This may happen if the user gives a simple acknowledgement (“yeah”) while the system is speaking, or if the speech was identified as noise (such as a cough).

As shown in Fig. 5, there is a feedback loop from the Vocalizer to the Contextualizer which informs the system of which concepts it has produced. The Contextualizer relates these concepts to the ones forwarded by the Interpreter. An example may help to clarify how this works. Fig. 9 shows an example where the system is about to confirm a

travel itinerary. At the top, the system's Speech Plan with associated concepts is shown (for the sake of simplicity, it contains no alternative paths). The realisation of the Speech Plan, and the user's reactions to it, is shown in the middle. At the bottom, the Context model, as produced by the Contextualizer, is shown. As soon as a Speech Segment has been realised, the Vocalizer adds timestamps to it, and the associated concept is forwarded to the Contextualizer. The Contextualizer can then follow the grounded-in links of the concepts and derive the time when they were realised. Since the user's concepts are grounded in the words in the Recognizer (which have timestamps), the Contextualizer can sort the user's and system's concepts according to when they were produced, and thereby accurately model how the concepts relate to each other in time. Fig. 9 (bottom) illustrates how the user's feedback to the different parts of the system's utterance is modelled. For simple acknowledgement like "yeah", the AM takes no specific action, which allows the Vocalizer to continue rendering the Speech Plan. However, when the user makes a correction ("no, four"), the AM decides to produce a new Speech Plan as a reaction to this (such as "ok, four o'clock"), and the old one is aborted. The fact that the date ("on the first of July"), which was part of the original Speech Plan, was never realised is now accurately modelled by the Contextualizer, which may trigger the AM to confirm the date as soon as the misunderstanding has been resolved.

In contrast to this, a system which does not self-monitor its own spoken output would have problems relating the user's feedback to the different parts of the itinerary. It would be problematic to determine what "yeah" is a reaction to, and whether "no, four" corrects the number of persons or the time of departure. It would also be problematic for the system to decide whether to stop speaking or continue, depending on whether the user made a positive acknowledgement or a correction. Finally, the system would not know which parts of the utterance would actually have been realised and acknowledged before the interruption. It would have to treat the whole utterance as either completely realised or basically unsaid.

It should be stressed that this kind of self-monitoring could also be done in a non-incremental system. However, incrementality makes the processing simpler, since the context model can be updated as the concepts are produced and thus needs less revision (which would be needed if large chunks of incremental units are received at the same time). Also, in the example above, the system can make a decision at every feedback place, to decide which feedback to give, given the current state of the context model. It should also be stressed that the contextual resolution in the example above (simply relating the feedback to the last concept) is much simplified. However, this is just an example that illustrates the benefits of incremental processing. The model itself does not hinder the contextualizer from doing a more sophisticated interpretation, trying to look at several possible "antecedents" for the feedback.

3.5. Self-repairs

As discussed previously, an incremental system must be open for revision (until points of commitment). On the output side, this means that a Speech Plan may be revised while it is being realised. To handle revisions in the spoken output, the Vocalizer keeps a list of the Speech Segments it has realised so far. If the Speech Plan is revised, the Vocalizer compares the history with the new graph and chooses to either do a *covert* or *overt* repair.

- **Covert repair:** If the history of realised segments matches a possible path through the new Speech Plan, the Vocalizer smoothly switches to the new plan without the user noticing it. The switch is done between two Speech Units.
- **Overt repair:** Even if the complete history of realised segments does not match a possible path through the new Speech Plan, the Vocalizer tries to match as much as possible and then continue the realisation from where the realisation and the new Speech Plan diverge. An editing term (e.g., "sorry") is also inserted, unless all segments that are to be repaired have the COMMITTING property set to false.

Fig. 10 illustrates both kinds of repairs in a travel booking domain. In this scenario, the ASR uses some time-consuming processing, such as a very wide search beam. Since ASR processing and endpoint detection operate asynchronously, the system may detect an end-of-turn and start to speak, even if the ASR processing is not complete. After t_3 , the system detects a relevant place to take the turn. At t_5 , the AM has produced a tentative, incomplete Speech Plan based on the incomplete processing of input. The Speech Plan starts with OPTIONAL floor-holders (marked with dotted outlines). Since no FINAL segment is in sight, the Vocalizer starts to realise such a segment. As more input is processed in t_7 , the AM detects that there are no available flights from London and the Speech Plan is amended accordingly to inform the user of this. When "London" is revised and replaced by "Londonderry" in t_8 , the AM

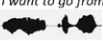




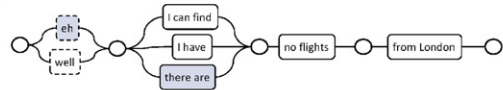

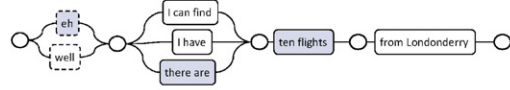

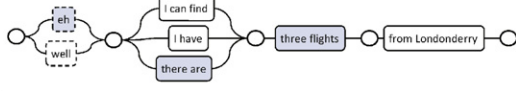

	ASR result and Speech Plan from AM	Speech channel
t1		I want to go from 
t2	ASR: I want	
t3		I want to go from Londonderry in the morning 
t4	ASR: I want to go	
t5		I want to go from Londonderry in the morning eh, there 
t6	ASR: I want to go from London	
t7		I want to go from Londonderry in the morning eh, there are 
t8	ASR: I want to go from Londonderry	
t9		I want to go from Londonderry in the morning eh, there are ten 
t10	ASR: I want to go from Londonderry in the morning	
t11		I want to go from Londonderry in the morning eh, there are ten, sorry, three flights 

Fig. 10. Overt and covert self-repairs during concurrent production and realisation of Speech Plans. The ASR results are delayed due to time-consuming processing.

discovers that there are available flights, and the Speech Plan is replaced. Since the realisation history in the Vocalizer (“eh, there are”) matches the new Speech Plan, a *covert* repair can be made in t_9 . As more input is processed in t_{10} , the query is constrained so that only morning flights are looked for, which results in a revision of the Speech Plan. Since the system has already started to produce the COMMITTING segment “ten flights”, which is not part of the new plan, the Vocalizer needs to insert an editing term (“sorry”) in order to continue with the new plan, i.e., an *overt repair* is made.

3.6. Implementation

The general, abstract model outlined in Section 3.1 has found several implementations, in the form of middleware for incremental systems (Schlangen et al., 2010). One of them is Jindigo (Skantze, 2010), which has been used to implement the model proposed in this paper. Jindigo provides a set of standard modules with which it is possible to implement incremental conversational systems. In Jindigo, IUs are modelled as typed objects, where all IUs are derived from a base class that handles grounded-in relations, same-level links, etc. Modules run asynchronously, i.e., update messages do not directly invoke methods in other modules, but are put on the input queues of the receiving modules. The update messages are then processed by each module in their own thread.

For speech recognition, we use CMU Sphinx 4 (Lamere et al., 2003), which has been adapted to act as a Jindigo module. We have implemented an Interpreter that is similar to Skantze and Edlund (2004), which robustly parses speech recognition results and produces conceptual tree structures. The Contextualizer operates similarly to the discourse modeller Galatea, as described in Skantze (2008).

For speech synthesis, we use MaryTTS (Schröder and Trouvain, 2003). MaryTTS does not support incremental speech synthesis, which means that complete Speech Segments must be synthesised before starting to realise them. To handle covert and overt repairs within a segment, the timing of the words in the synthesized segment are requested from MaryTTS and used to partition the audio into Speech Units (i.e., one unit per word). To find a more precise point

where the Vocalizer can pause or switch segments without any glitch, the Vocalizer searches for a zero-crossing point in the audio signal near the unit boundary. When the segment is realised, the Vocalizer feeds one unit at a time to the audio output device. This makes it possible to stop between two units and start to realise another segment, possibly starting from a mid-segment unit, resulting in a seamless switch.

The Speech Plan allows for a mix of pre-synthesised segments and segments which are synthesised “live”. For example, segments such as “eh” and “you can have it for” can be pre-synthesised with a (hand-crafted or data-driven) prosody that matches the intended conversational style, while segments such as “60 crowns” can be synthesised by the Vocalizer during the dialogue.

4. An experiment in a Wizard-of-Oz setting

In order to build a conversational system, data is needed on how users behave when interacting with the system. The problem is how to collect such data before the system is built. This chicken-and-egg problem is often dealt with by having a human operator simulating parts of the system, without the user being aware of it – a so-called Wizard-of-Oz setting (Fraser and Gilbert, 1991). The operator (Wizard) can simulate a single component (like the ASR) or more or less the complete system. As discussed in Section 2.1, a common problem with this kind of setting is the time it takes for the Wizard to manage the task (such as transcribe what the user is saying), which may result in long response delays. The Wizard-of-Oz setting therefore provides an interesting challenge for the incremental model of speech generation presented here.

4.1. DEAL – a conversational system for language learners

The system used in the experiment was a conversational system for second language learners of Swedish under development at KTH, called DEAL (Wik and Hjalmarsson, 2009). The scene of DEAL is set at a flea market where a talking agent is the owner of a shop selling used goods. The student is given a mission to buy items at the flea market getting the best possible price from the shopkeeper. The shopkeeper can talk about the properties of goods for sale and negotiate about the price. The price can be reduced if the user points out a flaw of an object, argues that something is too expensive, or offers lower bids. However, if the user is too persistent in haggling, the agent gets frustrated and closes the shop. Then the user has failed to complete the task. For the experiment, DEAL was re-implemented using the Jindigo framework. Fig. 11 shows the GUI that was shown to the user.

In this Wizard-of-Oz setup, the Wizard’s task was to transcribe the user’s speech, the rest of the system was completely automatic. In order to relieve the Wizard of any turn-taking decisions, a voice activity detector (VAD) was used. As soon as 0.5 s of silence was detected, the system initiated a response even if the Wizard had not completed the transcription yet. The setting is shown in Fig. 12 (compare with Fig. 5). The Wizard could start typing as soon as the user started to speak and could alter anything until the return key was pressed and the hypothesis was committed. As soon as the Wizard ended a word by typing a space, the word buffer was updated. Thus, the word buffer was updated on a word-by-word basis, similar to the output of an ASR.

For comparison, we also configured a non-incremental version of the same system. In this version of the system, nothing was sent from the Wizard until he committed by pressing the return key. Since we did not have mature models for the Interpreter, the Wizard was allowed to adapt the transcription of the utterances to match the models, while preserving the semantic content.

For the non-incremental version, there are of course several possibilities for coping with the delayed responses, in order to make a fair comparison with the incremental version. The system could for example start to play a jingle or ticking sound as soon as the user finished speaking (a common practice in commercial systems when responses are delayed (Balentine and Morgan, 2001)). However, since the domain of the system was a conversational game, we wanted to adhere to a human-like metaphor, and avoid non-conversational phenomena which could potentially confuse the users. Instead, the non-incremental version was simply silent until the response was ready.

In a previous data collection of human–human interaction in the DEAL domain (Hjalmarsson, 2008) it was noted that about 40% of the speaker turns were initiated with standardised lexical expressions such as “ja” (Eng: “yes”), “eh” (Eng: “eh”), and “vänta lite” (Eng: “wait a minute”). Such speech segments commit very little semantically to the message and are therefore very useful as initiations before the system has planned a complete response. The system

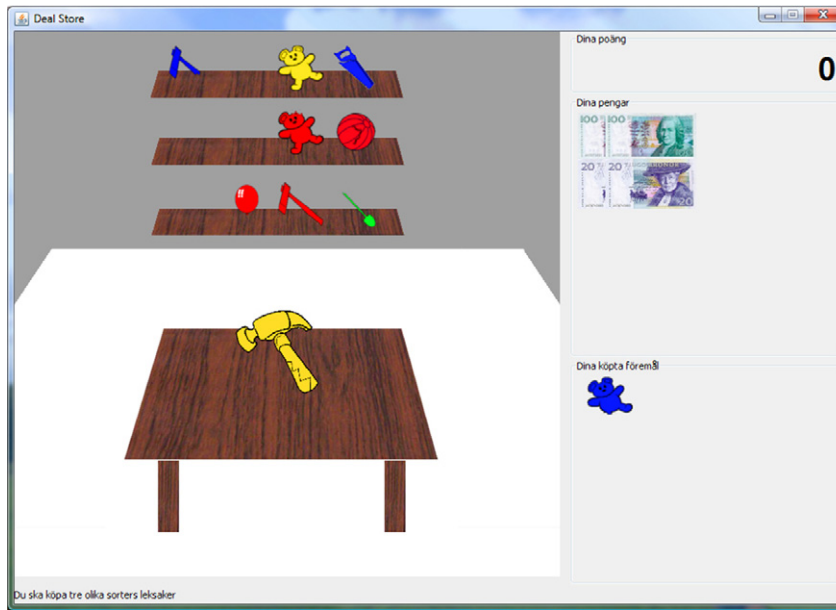


Fig. 11. The user interface in DEAL. The object on the table is the one currently in focus. Example objects are shown on the shelf. Current game score, money and bought objects are shown on the right.

can start to produce these expressions immediately after the user has stopped speaking, allowing the Wizard to exploit the additional time to transcribe what the user said.

The DEAL corpus was used to create initial speech segments for the experiment. The motivation to use speech segments derived from human recordings was to make the system sound convincing in terms of both lexical choice and intonation. In particular, we wanted a repertoire of different types of filled pauses and feedback expression such as “eh” and “mm” in order to avoid a system that sounds monotone and repetitive. First, a number of feedback expressions such as “ja”, “a”, “mm” (Eng: “yes”), filled pauses such as “eh”, “ehm” and expressions used to initiate different domain specific speech acts (e.g. “it costs...” and “here is a...”) were extracted. The segments were re-synthesized using Expros, a tool for experimentation with prosody in diphone voices (Gustafson and Edlund, 2008). Based on manual transcriptions and sound files, Expros automatically extracts pitch, duration and intensity from the human voice and creates a synthetic version using these parameters. In the speech plan, these “canned” segments were mixed with text

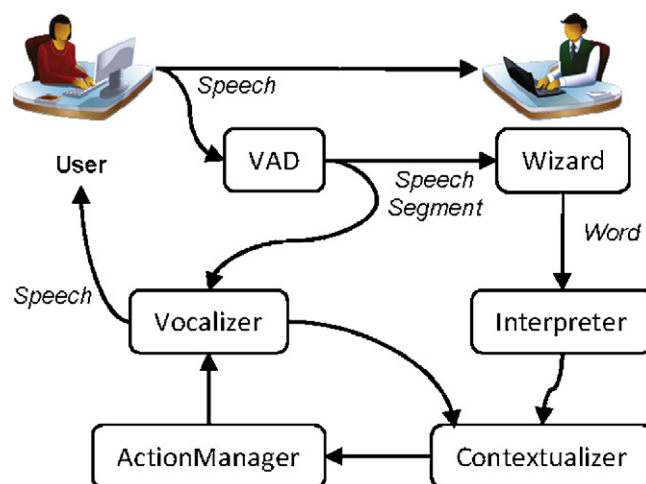


Fig. 12. The system architecture used in the Wizard-of-Oz experiment.

Table 1

An example DEAL dialogue (translated from Swedish). Speech Segments are marked in brackets.

S.1	[welcome] [how may I help you]
U.2	<i>I want to buy a doll</i>
S.3	[eh] [here is] [a doll]
U.4	<i>how much is it?</i>
S.5	[eh] [it costs] [120 crowns]
U.6	<i>that is too expensive how much is the teddy bear?</i>
S.7	[well] [you can have it for] [let's see] [40 crowns]
U.8	<i>I can give you 30 crowns</i>
S.9	[you could have it for] [37 crowns]
U.10	<i>I can give you 10 crowns</i>
S.11	[let's say] [or, I mean] [that is way too little]

segments (for example references to objects, prices, etc.) that were generated and synthesized on-line with the same diphone voice.

An example interaction with the incremental version of the system is shown in Table 1. S.11 exemplifies a self-correction, where the system prepares to present another bid, but then realises that the user's bid is too low to even consider. A video (with subtitles) showing an interaction with one of the users can be seen at <http://www.youtube.com/watch?v=cQQmgItIMvs>.

4.2. Experimental setup

In order to compare the incremental and non-incremental versions of the system, we conducted an experiment with 10 participants, 4 male and 6 female. The participants were given the task of buying three items (with certain characteristics) from the shop-keeper in DEAL at the best possible price. The participants were further instructed to evaluate two different versions of the system: System A and System B. However, they were not informed how the versions differed. The participants were lead to believe that they were interacting with a fully working conversational system and were not aware of the Wizard-of-Oz setup. Each participant interacted with the system four times; first, they interacted one time with each version of the system; second, they completed a questionnaire; third, they interacted with each version again; and finally, they filled out the questionnaire again. The order of the system versions was balanced between subjects.

The mid-experiment questionnaire was used to collect the participants' first opinions of the two versions of the DEAL system and to make them aware of what type of characteristics they should consider when interacting with the system the second time. When filling out the second questionnaire, the participants were asked to base their ratings on their overall experience with the two system versions. Thus, the analysis of the results is based on the second questionnaire. In the questionnaires, they were requested to rate which one of the two versions was most prominent according to 8 different dimensions: which version they preferred; which was more human-like, polite, efficient, and intelligent; which gave a faster response and better feedback; and with which version it was easier to know when to speak. All ratings were done on a continuous horizontal line with System A on the left end and System B on the right end. The centre of the line was labelled with "no difference".

The participants were recorded during their interaction with the system, and all system messages were logged.

4.3. Results

Fig. 13 shows the difference in response time between the two versions. As expected, the incremental version started to speak more quickly ($M = 0.55$ s, $SD = 1.25$) than the non-incremental version ($M = 2.85$ s, $SD = 1.15$; $t(702) = 25.45$, $p < .001$), while producing longer utterances ($M = 4.47$ s, $SD = 1.73$ vs. $M = 2.81$ s, $SD = 1.25$; $t(650) = 14.64$, $p < .001$). It was harder to anticipate whether it would take more or less time for the incremental version to finish utterances. Both versions received the final input at the same time. On the one hand, the incremental version initiates utterances with speech segments that contain little or no semantic information. Thus, if the system is in the middle of such a segment when receiving the complete input from the Wizard, the system may need to complete this segment before producing

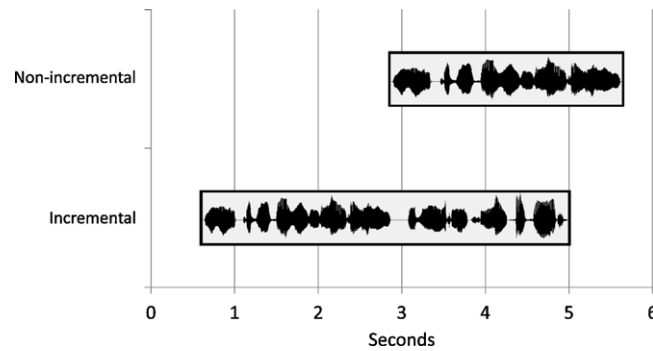


Fig. 13. The average response time and length of the system's response, counted from the end of the preceding user utterance.

the rest of the utterance. Moreover, if an utterance is initiated and the Wizard alters the input, the incremental version needs to make a repair which takes additional time. On the other hand, it can also start to produce speech segments that are semantically relevant, based on the incremental input, which allows it to finish the utterance more quickly. As the figure shows, it turns out that the average response completion time for the incremental version ($M = 5.02$ s, $SD = 1.54$) is about 600 ms faster than the average for the non-incremental version ($M = 5.66$ s, $SD = 1.50$), ($t(704) = 5.56$, $p < .001$). This shows that the incremental version indeed managed to produce meaningful segments while the Wizard was typing, and that this saved more time than was lost due to fillers and self-repairs.

In general, subjects reported that the system worked very well. After the first interaction with the two versions, the participants found it hard to point out differences, as they were focused on solving the task. The marks on the horizontal continuous lines on the questionnaire were measured with a ruler based on their distance from the midpoint (labelled with “no difference”) and normalized to a scale from -1 to 1 , each extreme representing one system version. A Wilcoxon Signed Ranks Test was carried out, using these rankings as differences. The results are shown in Fig. 14. As the figure shows, the two versions differed significantly in three dimensions, all in favour of the incremental version. Hence, the incremental version was rated as more polite, more efficient, and better at indicating when to speak.

While listening to the recordings of the dialogues it was noted that the users sometimes repeated information or made a new request before the non-incremental version had time to respond to their initial request. In order to explore this further, all such events were manually annotated. Only instances where the user was silent for at least 0.2 s before speaking again were annotated. The reason for this was to avoid immediate alterations which could not possibly have been reactions to the system's response delay. Thus, these immediate alterations were more likely a result of a change of mind. The annotated repetitions were all instances where the user repeated all or parts of the information before the

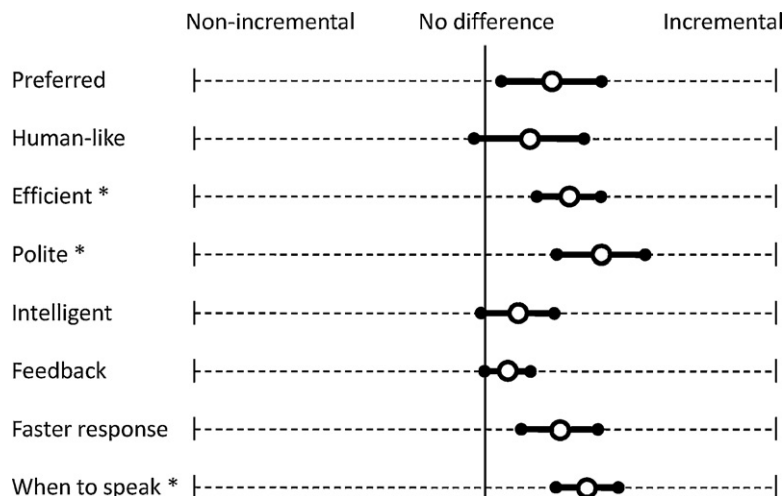


Fig. 14. The results from the second questionnaire. The graph shows the mean scores (unfilled circles) together with the standard errors (filled circles). Significant differences between the two systems are marked with *.

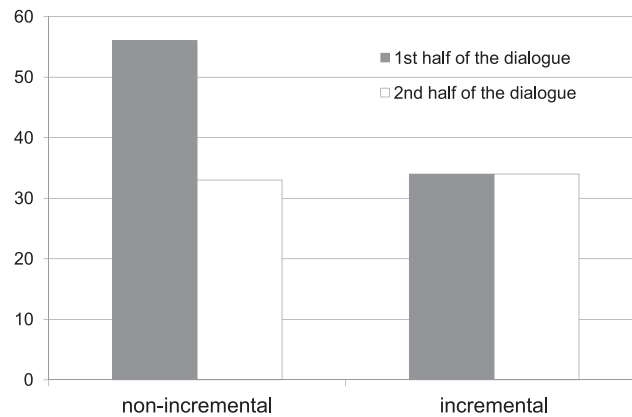


Fig. 15. The number of restarts and repetitions in the first and the second half of the dialogues distributed over the incremental and the non-incremental system.

system had responded to the first request (e.g. “can I have a blue doll?” [0.2 s < silence] “do you have a blue doll?”). Restarts were when the user made a new request before the system had responded to the user’s previous request (e.g. “can I have a blue doll?” [0.2 s < silence] “can I have a green watch?”). The annotator was not aware of which system version the user was interacting with and in order to avoid revealing this, the annotations were done without listening to the systems’ responses. The distribution of repetitions and restarts are presented in Fig. 15.

A Chi-square Goodness-of-Fit test was employed to investigate whether the distribution of user restarts and repetitions differed between the incremental and the non-incremental versions of the system. The dialogues were split into two equally long halves based on the total duration of each dialogue. The results show that there is a significant difference ($p < .05$) in the number of restarts and repetitions between the four categories ($p = .02$, $n = 157$, $df = 3$, $\chi^2 = 9.55$). To determine which cells in the cross-tabulation had higher than expected frequencies, the standard residual for each cell was examined. The results suggest that the number of restarts and repetitions during the first half of the dialogues with the non-incremental version is significantly higher than expected frequency.

The analysis suggests that the non-incremental version’s long response times were confusing to the users. Compared to when interacting with the incremental system, the users used significantly more repetitions and restarts when talking to the non-incremental version. It is possible that they did not know whether the system had perceived what they said and therefore made a second request. Evidently, since the incremental version initiated responses based on a silence threshold, there was less possibility for the user to pause and speak again. Still, during the second halves of the dialogues, there were no difference between the two versions. Thus, during the latter parts of the dialogues, the users

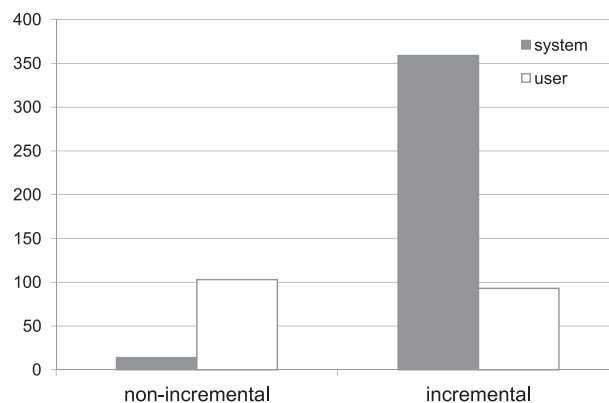


Fig. 16. The number of system and user fillers distributed over the incremental and the non-incremental system.

appear to have adjusted to the non-incremental version's response delay and the number of restarts and repetitions used were very similar to those of the incremental version.

Another interesting aspect of the experiment is whether the users were influenced by the two system versions and behaved differently when interacting with them. A well-known phenomena in dialogue is that of entrainment (or adaptation or alignment), that is, speakers (in both human–human and human–computer dialogue) tend to adapt the conversational behaviour to their interlocutor (see for example [Bell, 2003](#)). In order to examine whether the different versions affected the users' behaviour, we analyzed both the users' response time and the number of fillers that the users produced, but found no significant differences between the interactions with the two versions. [Fig. 16](#) shows the number of user and system fillers for the two different system versions. While the incremental system produces significantly more fillers than the non-incremental system version, there is no difference in the number of fillers produced by the users. Thus, it might be the case that fillers and response time are behaviours that to a larger extent are affected by other factors such as interactional and cognitive constraints. Another possible explanation is that the users switched back and forth between the two versions and therefore did not spend enough time with either system in order to entrain.

5. Conclusions and future work

This paper has presented a first step towards incremental speech generation in conversational systems. First, we presented an incremental model for speech generation that starts to plan a response while the user is still speaking. When the system detects a relevant place to take the turn, it asynchronously realises this plan while processing continues, with the possibility to revise the plan if needed. In order to test the utility of this model, we conducted an experiment with a dialogue system using a Wizard-of-Oz setting. The results are promising: while producing longer utterances, the incremental dialogue system starts to speak faster and produces a complete response before the non-incremental version. Analyses of user repetitions and restarts suggest that these behaviours were more frequent during the first half of the dialogues with the non-incremental version. However, during the second part the users appear to have adjusted to this response delay and wait for the system to respond to their initial request. This change in behaviour suggests that users are capable of adjusting to the system's somewhat slow turn-taking behaviour. However, this adjustment likely comes with an extra cognitive burden. Furthermore, compared to the non-incremental system, the incremental version was rated as more polite, more efficient, and better at indicating when to speak.

As this is a first step, there are several ways to improve the model. First, the system could be made more efficient time wise when choosing Speech Segments while waiting for the final Speech Segment. For example, when the user has finished speaking, it should (in some cases) be possible to anticipate how long it will take until processing is completed and thereby choose a more optimal path through the Speech Plan (by taking the length of the Speech Segments into consideration). A statistical model could be interesting to explore for this purpose. There is also the option of not starting to produce speech in the form of filled pauses, but also to sometimes just wait with the response. By increasing the *OtherDelay* parameter of the initial filler speech segments, the Vocalizer would automatically produce more silence and select the fillers less often (since the final segments would be ready more often when the system starts to speak). If the system can detect different types of turn-taking behaviours (e.g., different types of lexical, prosodic and visual features) in order to make more sophisticated turn-taking decisions, these features can be used to calculate the probability of a turn-ending and based on this probability vary the response time dynamically, similar to [Raux and Eskenazi \(2008\)](#).

The system could also be made more efficient in its self-repair strategies. For example, in order to avoid a lot of self-repairs, the Action Manager should produce less committing segments early in the plan and present the more committing segments at the end. A possible future direction could be to let the system learn by previous interactions how responses to certain kinds of questions typically start. For example, the system might discover that answers to requests that starts with “how much is...” very often starts with “it costs”. This would allow the system to produce Speech Segments that provide semantically relevant information early on in the plan in order to save time, without having to risk a time-consuming self-repair.

There are also other possible applications of incremental speech generation, besides conversational systems. One such application is computer-mediated human–human interaction for persons who have problems speaking. This could include persons with speech disorders, but also a person who is unable to speak due to the circumstances, for example because she is in a meeting. By using some kind of input device, such as a keyboard, an incremental speech synthesizer could transform the manual input into speech as the person is typing. Thus, the person could engage in an online conversation (for example over the telephone), without speaking. The system could automatically insert filled pauses

in order to keep the floor when the user needs additional time for typing, and make self-corrections if the persons edits the input to the system.

Finally, the experiment shows that it is possible to produce fast responses in a Wizard-of-Oz setting where the Wizard takes time to respond. This opens up new possibilities for the Wizard-of-Oz paradigm, and thereby for practical development of conversational systems in general.

Acknowledgements

This research was funded by the Swedish Research Council (VR) projects *Incremental processing in multimodal conversational systems* (#2011-6237) and *Classifying and deploying pauses for flow control in conversational systems* (#2011-6152). We would also like to thank Joakim Gustafson for rewarding discussion and ideas.

References

- Allen, J.F., Ferguson, G., Stent, A., 2001. An architecture for more realistic conversational systems. In: Proceedings of the 6th International Conference on Intelligent User Interfaces (IUI-01), Santa Fe, NM, January, pp. 1–8.
- Balentine, B., Morgan, D.P., 2001. How to Build a Speech Recognition Application: A Style Guide for Telephony Dialogues, 2nd edition. Enterprise Integration Group, San Ramon, CA.
- Bell, L., 2003. Linguistic adaptations in spoken human–computer dialogues: empirical studies of user behavior. Ph.D. thesis. KTH, Stockholm, Sweden.
- Blackmer, E.R., Mitton, J.L., 1991. Theories of monitoring and the timing of repairs in spontaneous speech. *Cognition* 39 (3), 173–194.
- Brysbaert, M., Fias, W., 1998. The Whorfian hypothesis and numerical cognition: is “twenty-four” processed in the same way as “four-and-twenty”? *Cognition* 66 (1), 51–77.
- Caramazza, A., 1997. How many levels of processing are there in lexical access? *Cognitive Neuropsychology* 14 (1), 177–208.
- DeVault, D., Sagae, K., Traum, D., 2009. Can I finish? Learning when to respond to incremental interpretation results in interactive dialogue. In: Proceedings of SigDial, London, UK, pp. 11–20.
- Dohsaka, K., Shimazu, A., 1997. System architecture for spoken utterance production in collaborative dialogue. In: Working Notes of IJCAI 1997 Workshop on Collaboration. Cooperation and Conflict in Dialogue Systems, Nagoya, Japan.
- Ferreira, F., Swets, B., 2002. How incremental is language production? Evidence from the production of utterances requiring the computation of arithmetic sums. *Journal of Memory and Language* 46 (1), 57–84.
- Fraser, N., Gilbert, N., 1991. Effects of system voice quality on user utterances in speech dialogue systems. In: Proceedings of Eurospeech, Genova, Italy, pp. 57–60.
- Fromkin, V., 1973. *Speech Errors as Linguistic Evidence*. The Hague, Mouton, New York, NY.
- Gravano, A., Hirschberg, J., Sep 2009. Turn-yielding cues in task-oriented dialogue. In: Proceedings of SigDial, London, UK, pp. 253–261.
- Guhe, M., 2007. *Incremental Conceptualization for Language Production*. Lawrence Erlbaum Associates, Mahwah, NJ.
- Gustafson, J., Edlund, J., 2008. Expros: a toolkit for exploratory experimentation with prosody in customized diphone voices. In: Proceedings of Perception and Interactive Technologies for Speech-Based Systems (PIT 2008). Kloster Irsee, Germany, pp. 293–296.
- Heintze, S., Baumann, T., Schlangen, D., 2010. Comparing local and sequential models for statistical incremental natural language understanding. In: Proceedings of SigDial, Tokyo, Japan, pp. 9–16.
- Hjalmarsson, A., 2008. Speaking without knowing what to say... or when to end. In: Proceedings of SigDial, Columbus, OH, pp. 72–75.
- Howell, P., Young, K., 1991. The use of prosody in highlighting alterations in repairs from unrestricted speech. *The Quarterly Journal of Experimental Psychology* 43a (3), 733–758.
- Kempen, G., Hoenkamp, E., 1982. Incremental sentence generation: implications for the structure of a syntactic processor. In: Proceedings of COLING, Prague, Czech Republic, pp. 151–156.
- Kempen, G., Hoenkamp, E., 1987. An incremental procedural grammar for sentence formulation. *Cognitive Science* 11 (2), 201–258.
- Kilger, A., Finkler, W., 1995. Incremental generation for real-time applications. Research Report RR-95-11, DFKI, Saarbrücken, Germany.
- Lamere, P., Kwok, P., Gouvea, E., Raj, B., Singh, R., Walker, W., Warmuth, M., Wolf, P., 2003. The CMU Sphinx-4 speech recognition system. In: Proceedings of ICASSP, Hong Kong, China, April.
- Levelt, W.J.M., 1989. *Speaking: From Intention to Articulation*. MIT Press, Cambridge, MA.
- Postma, A., 2000. Detection of errors during speech production: a review of speech monitoring models. *Cognition* 77 (2), 97–132.
- Raux, A., Eskenazi, M., 2008. Optimizing endpointing thresholds using dialogue features in a spoken dialogue system. In: Proceedings of SIGdial, Columbus, OH.
- Schlangen, D., Baumann, T., Buschmeier, H., Buss, O., Kopp, S., Skantze, G., Yaghoubzadeh, R., 2010. Middleware for incremental processing in conversational agents. In: Proceedings of SigDial, Tokyo, Japan.
- Schlangen, D., Skantze, G., 2011. A general, abstract model of incremental dialogue processing. *Dialogue and Discourse* 2 (April (1)), 83–111.
- Schröder, M., Trouvain, J., 2003. The German text-to-speech synthesis system Mary: a tool for research, development and teaching. *International Journal of Speech Technology* (6), 365–377.
- Shriberg, E., 1994. Preliminaries to a theory of speech disfluencies. Ph.D. thesis. University of California, Berkeley.

- Skantze, G., November, 2007. Error handling in spoken dialogue systems – managing uncertainty, grounding and miscommunication. Ph.D. thesis. Department of Speech, Music and Hearing, KTH.
- Skantze, G., August, 2008. Galatea: A Discourse Modeller Supporting Concept-Level Error Handling in Spoken Dialogue Systems. Springer.
- Skantze, G., 2010. Jindigo: a Java-based framework for incremental dialogue systems. Tech. rep. KTH, Stockholm, Sweden.
- Skantze, G., Edlund, J., 2004. Robust interpretation in the Higgins spoken dialogue system. In: Proceedings of ISCA Tutorial and Research Workshop (ITRW) on Robustness Issues in Conversational Interaction, Norwich, UK.
- Skantze, G., Schlangen, D., 2009. Incremental dialogue processing in a micro-domain. In: Proceedings of EACL, Athens, Greece, pp. 745–753.
- Wik, P., Hjalmarsson, A., 2009. Embodied conversational agents in computer assisted language learning. *Speech Communication* 51 (10), 1024–1037.