Towards Incremental Speech Generation in Dialogue Systems

Gabriel Skantze
Dept. of Speech Music and Hearing
KTH, Stockholm, Sweden
gabriel@speech.kth.se

Anna Hjalmarsson
Dept. of Speech Music and Hearing
KTH, Stockholm, Sweden
annah@speech.kth.se

Abstract

We present a first step towards a model of speech generation for incremental dialogue systems. The model allows a dialogue system to incrementally interpret spoken input, while simultaneously planning, realising and self-monitoring the system response. The model has been implemented in a general dialogue system framework. Using this framework, we have implemented a specific application and tested it in a Wizard-of-Oz setting, comparing it with a non-incremental version of the same system. The results show that the incremental version, while producing longer utterances, has a shorter response time and is perceived as more efficient by the users.

1 Introduction

Speakers in dialogue produce speech in a piecemeal fashion and on-line 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 realize new dialogue contributions. Moreover, interlocutors continuously self-monitor the actual production processes in order to facilitate self-corrections (Levelt, 1989). Contrary to this, most spoken dialogue systems use a silence threshold to determine when the user has stopped speaking. The user utterance is then processed by one module at a time, after which a complete system utterance is produced and realised by a speech synthesizer.

This paper has two purposes. First, to present an initial step towards a model of speech generation that allows a dialogue system to incrementally interpret spoken input, while simultaneously planning, realising and self-monitoring the system response. The model has been implemented in a general dialogue system framework. This is described in Section 2 and 3. The second purpose is to evaluate the usefulness of incremental speech generation in a Wizard-of-Oz setting, using the proposed model. This is described in Section 4.

1.1 Motivation

A non-incremental dialogue system waits until the user has stopped speaking (using a silence threshold to determine this) before starting to process the utterance and then produce a system response. If processing takes time, for example because an external resource is being accessed, this may result in a confusing response delay. An incremental system may instead continuously build a tentative plan of what to say as the user is speaking. When it detects that the user's utterance has ended, it may start to asynchronously realise this plan while processing continues, with the possibility to revise the plan if needed.

There are many potential reasons for why dialogue systems may need 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). In this paper, we will explore incremental speech generation in a Wizard-of-oz setting. 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 & 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.

Proceedings of SIGDIAL 2010: the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 1–8,
The University of Tokyo, September 24-25, 2010. ©2010 Association for Computational Linguistics

1.2 Related work

Incremental speech generation has been studied from different perspectives. From a psycholinguistic perspective, Levelt (1989) and others have studied how speakers incrementally produce utterances while *self-monitoring* the output, both overtly (listening to oneself speaking) and covertly (mentally monitoring what is about to be said). As deviations from the desired output is detected, the speaker may initiate *self-repairs*. If the item to be repaired has already been spoken, an *overt* repair is needed (for example by using an editing term, such as “sorry”). If not, the utterance plan may be altered to accommodate the repair, a so-called *covert* repair. Central to the concept of incremental speech generation is that the realization of overt speech can be initiated before the speaker has a complete plan of what to say. An option for a speaker who does not know what to say (but wants to claim the floor) is to use hesitation phenomena such as *filled pauses* (“eh”) or *cue phrases* such as “let’s see”.

A dialogue system may not need to self-monitor its output for the same reasons as humans do. For example, there is no risk of articulatory errors (with current speech synthesis technology). However, a dialogue system may utilize the same mechanisms of self-repair and hesitation phenomena to simultaneously plan and realise the spoken output, as there is always a risk for revision in the input to an incremental module (as described in Section 2.1).

There is also another aspect of self-monitoring that is important for dialogue systems. In a system with modules operating asynchronously, the dialogue manager cannot know whether the intended output is actually realized, as the user may interrupt the system. Also, the timing of the synthesized speech is important, as the user may give feedback in the middle of a system utterance. Thus, an incremental, asynchronous system somehow needs to self-monitor its own output.

From a syntactic perspective, Kempen & Hoenkamp (1987) and Kilger & Finkler (1995) have studied how to syntactically formulate sentences incrementally under time constraints. Dohsaka & Shimazu (1997) describes a system architecture for incremental speech generation. However, there is no account for revision of the input (as discussed in Section 2.1) and there is no evaluation with users. Skantze & Schlangen (2009) describe an incremental system that partly supports incremental output and that is evaluated with users, but the domain is limited to number dictation.

In this study, the focus is not on syntactic construction of utterances, but on how to build practical incremental dialogue systems within limited domains that can handle revisions and produce convincing, flexible and varied speech output in on-line interaction with users.

2 The Jindigo framework

The proposed model has been implemented in Jindigo – a Java-based open source framework for implementing and experimenting with incremental dialogue systems (www.jindigo.net). We will here briefly describe this framework and the model of incremental dialogue processing that it is based on.

2.1 Incremental units

Schlangen & Skantze (2009) describes a general, abstract model of incremental dialogue processing, which Jindigo is based on. In this model, a system consists of a network of processing modules. Each module has a left buffer, a processor, and a right buffer, where 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 example is shown in Figure 1. 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, etc). 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 below, IU_2 has a same-level link to IU_1 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, IU_3 is grounded in IU_1 and IU_2, while IU_4 is grounded in IU_3.

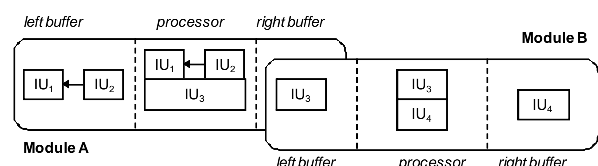

Figure 1: Two connected modules.

String	Right buffer	Update message
t_1: one	w1 ← one ← w2	[w1, w2]
t_2: one five	w1 ← one ← w2 ← five ← w3	[w1, w3]
t_3: one	w1 ← one ← w2 ← five ← w3	[w1, w2]
t_4: one four five	w1 ← one ← w2 ← five ← w3 w2 ← four ← w4 ← five ← w5	[w1, w5]
t_5: [commit]	w1 ← one ← w2 ← five ← w3 w2 ← four ← w4 ← five ← w5	[w5,w5]

Table 1: The right buffer of an ASR module, and update messages at different time-steps.

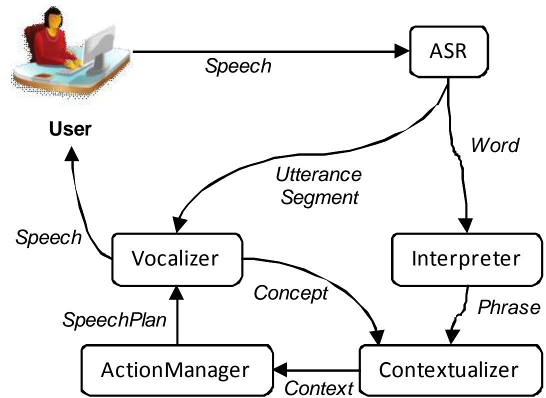

Figure 2: A typical Jindigo system architecture.

A challenge for incremental systems is to handle *revisions*. For example, as the first part of the word “forty” is recognised, the best hypothesis might be “four”. As the speech recogniser receives more input, it might need to revise its previous output, which might cause a chain of revisions in all subsequent modules. To cope with this, modules have to be able to react to three basic situations: that IUs are *added* to a buffer, which triggers processing; that IUs that were erroneously hypothesized by an earlier module are *revoked*, which may trigger a revision of a module’s own output; and that modules signal that they *commit* to an IU, that is, won’t revoke it anymore.

Jindigo implements an efficient model for communicating these updates. In this model, IUs are associated with edges in a graph, as shown in Table 1. The graph may be incrementally amended without actually removing edges or vertices, even if revision occurs. At each time-step, a new update message is sent to the consuming module. The update message contains a pair of pointers [*C*, *A*]: (*C*) the vertex from which the currently committed hypothesis can be constructed, and (*A*) the vertex from which the currently best tentative hypothesis can be constructed. In Jindigo, all modules run as threads within a single Java process, and therefore have access to the same memory space.

2.2 A typical architecture

A typical Jindigo system architecture is shown in Figure 2. The word buffer from the Recognizer module is parsed by the Interpreter module which tries to find an optimal sequence of top phrases and their semantic representations. These phrases are then interpreted in light of the current dialogue context by the Contextualizer module and are packaged as Communicative Acts (CAs). As can be seen in Figure 2, the Contextualizer also self-monitors Concepts from the system as they are spoken by the Vocalizer, which makes it possible to contextually interpret user responses to system utterances. This also makes it possible for the system to know whether an intended utterance actually was produced, or if it was interrupted. The current context is sent to the Action Manager, which generates a SpeechPlan that is sent to the Vocalizer. This is described in detail in the next section.

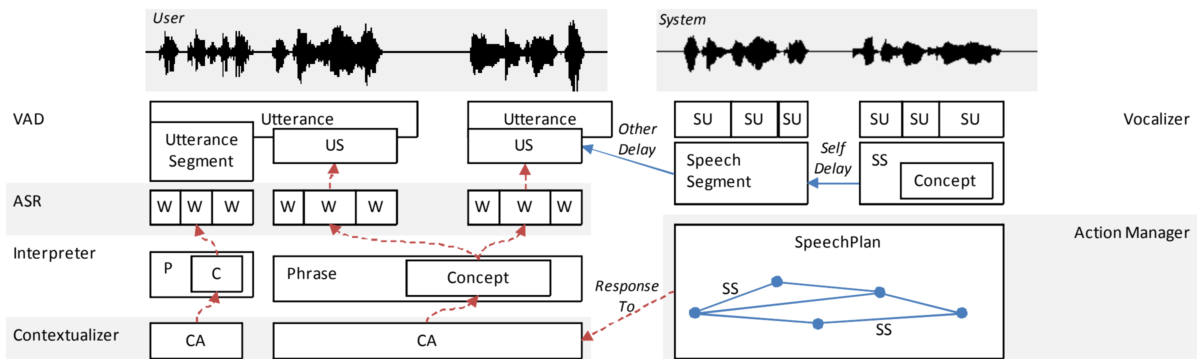

Figure 3: Incremental Units at different levels of processing. Some grounded-in relations are shown with dotted lines. W=Word, SS=SpeechSegment, SU=SpeechUnit, CA=Communicative Act.

3 Incremental speech generation

3.1 Incremental units of speech

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 Figure 3. Using a standard voice activity detector (VAD) in the ASR, the user's speech is chunked into **Utterance**-units. The Utterance boundaries determine when the ASR hypothesis is committed. However, for the system to be able to respond quickly, the end silence threshold of these Utterances are typically too long. Therefore smaller units of the type **UtteranceSegment** (US) are detected, using a much shorter silence threshold of about 50ms. Such short silence thresholds allow the system to give very fast responses (such as backchannels). Information about US boundaries is sent directly from the ASR to the Vocalizer. As Figure 3 illustrates, the grounded-in links can be followed to derive the timing of IUs at different levels of processing.

The system output is also modelled using IUs at different processing levels. The widest-spanning IU on the output side is the **SpeechPlan**. The rendering of a SpeechPlan will result in a sequence of **SpeechSegment**'s, where each SpeechSegment represents a continuous audio rendering of speech, either as a synthesised string or a pre-recorded audio file. For example, the plan may be to say "okay, a red doll, here is a nice doll", consisting of three segments. Now, there are two requirements that we need to meet. First, the output should be *varied*: the system should not give exactly the same response every time to the same request. But, as we will see, the output in an incremental system must also be *flexible*, as speech plans are incrementally produced and amended. In order to relieve the Action Manager of the burden of varying the output and making time-critical adjustments, we model the SpeechPlan as a directed graph, where each edge is associated with a SpeechSegment, as shown in Figure 4. Thus, the Action Manager may asynchronously plan (a set of possible) responses, while the Vocalizer selects the rendering path in the graph and takes care of time-critical synchronization. To control the rendering, each SpeechSegment has the properties optional, committing, selfDelay and otherDelay, as described in the next section. It must also be possible for an incremental system to interrupt and make self-repairs in the middle of a SpeechSegment. Therefore, each SpeechSegment may also be decomposed into an array of **SpeechUnit**'s, where each SpeechUnit contains pointers to the audio rendering in the SpeechSegment.

3.2 Producing and consuming SpeechPlans

The SpeechPlan does not need to be complete before the system starts to speak. An example of this is shown in Figure 4. As more words are recognised by the ASR, the Action Manager may add more SpeechSegment's to the graph. Thus, the system may start to say "it costs" before it knows which object is being talked about.

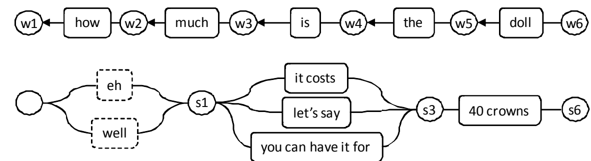

Figure 4: The right buffer of an ASR (top) and the SpeechPlan that is incrementally produced (bottom). Vertex s1 is associated with w1, s3 with w3, etc. Optional, non-committing SpeechSegment's are marked with dashed outline.

The SpeechPlan has a pointer called finalVertex. When the Vocalizer reaches the finalVertex, the SpeechPlan is completely realised. If finalVertex is not set, it means that the SpeechPlan is not yet completely constructed. The SpeechSegment property optional tells whether the segment needs to be realised or if it could be skipped if the finalVertex is in sight. This makes it possible to insert floor-keeping SpeechSegment's (such as "eh") in the graph, which are only realised if needed. The Vocalizer also keeps track of which SpeechSegment's it has realised before, so that it can look ahead in the graph and realise a more varied output. Each SpeechSegment may carry a semantic representation of the segment (a Concept). This is sent by the Vocalizer to the Contextualizer as soon as the segment has been realised.

The SpeechSegment properties selfDelay and otherDelay regulate the timing of the output (as illustrated in Figure 3). They specify the number of milliseconds that should pass before the Vocalizer starts to play the segment, depending on the previous speaker. By setting the otherDelay of a segment, the Action Manager may delay the response depending on how certain it is that it is appropriate to speak, for example by considering pitch and semantic completeness. (See Raux & Eskenazi (2008) for a study

on how such dynamic delays can be derived using machine learning.)

If the user starts to speak (i.e., a new `UtteranceSegment` is initiated) as the system is speaking, the Vocalizer pauses (at a `SpeechUnit` boundary) and waits until it has received a new response from the Action Manager. The Action Manager may 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, for example, the `UtteranceSegment` was identified as a back-channel, cough, or similar.

3.3 Self-repairs

As Figure 3 shows, a `SpeechPlan` may be grounded in a user CA (i.e., it is a response to this CA). If this CA is revoked, or if the `SpeechPlan` is revised, the Vocalizer may initialize a self-repair. The Vocalizer keeps a list of the `SpeechSegment`'s it has realised so far. If the `SpeechPlan` is revised when it has been partly realised, the Vocalizer compares the history with the new graph and chooses one of the different repair strategies shown in Table 2. In the best case, it may smoothly switch to the new plan without the user noticing it (covert repair). In case of a unit repair, the Vocalizer searches for a zero-crossing point in the audio segment, close to the boundary pointed out by the `SpeechUnit`.

covert segment repair	you are right → it is blue you are right → they are blue
overt segment repair	you are right → it is blue sorry → you are wrong → it is red
covert unit repair	you are right → it is blue you are wrong → it is red
overt unit repair	you are right → it is blue sorry you are wrong → it is red

Table 2: Different types of self-repairs. The shaded boxes show which `SpeechUnit`'s have been realised, or are about to be realised, at the point of revision.

The `SpeechSegment` property `committing` tells whether it needs to be repaired if the `SpeechPlan` is revised. For example, a filled pause such as "eh" is not committing (there is no need to insert an editing term after it), while a request or an assertion usually is. If (parts of) a committing segment has already been realised and it cannot be part of the new plan, an overt repaired is made with the help of an editing term (e.g., "sorry"). When comparing the history with the new graph, the Vocalizer searches the graph and tries to find a path so that it may avoid making an overt repair. For example if the graph in Figure 4 is replaced with a corresponding one that ends with "60 crowns", and it has so far partly realised "it costs", it may choose the corresponding path in the new `SpeechPlan`, making a covert repair.

4 A Wizard-of-Oz experiment

A Wizard-of-Oz experiment was conducted to test the usefulness of the model outlined above. All modules in the system were fully functional, except for the ASR, since not enough data had been collected to build language models. Thus, instead of using ASR, the users' speech was transcribed by a Wizard. As discussed in section 1.1, a common problem is the time it takes for the Wizard to transcribe incoming utterances, and thus for the system to respond. Therefore, this is an interesting test-case for our model. In order to let the system respond as soon as the user finished speaking, even if the Wizard hasn't completed the transcription yet, a VAD is used. The setting is shown in Figure 5 (compare with Figure 2). The Wizard may start to type as soon as the user starts to speak and may alter whatever he has typed until the return key is pressed and the hypothesis is committed. The word buffer is updated in exactly the same manner as if it had been the output of an ASR.

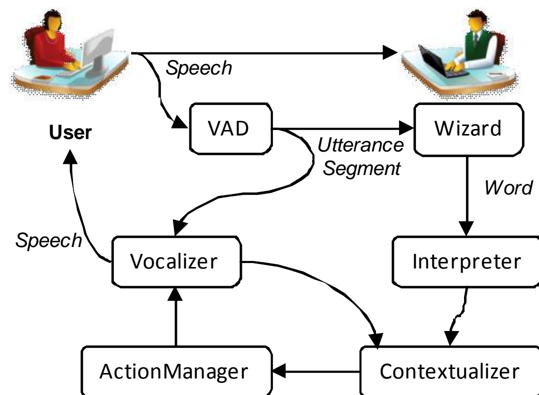

Figure 5: The system architecture used in the Wizard-of-Oz experiment.

For comparison, we also configured a non-incremental version of the same system, where nothing was sent from the Wizard until he com-

mitted by pressing the return key. Since we did not have mature models for the Interpreter either, the Wizard was allowed to adapt the transcription of the utterances to match the models, while preserving the semantic content.

4.1 The DEAL domain

The system that was used in the experiment was a spoken dialogue system for second language learners of Swedish under development at KTH, called DEAL (Hjalmarsson et al., 2007). 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 shop-keeper. The shop-keeper 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 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. Figure 6 shows the GUI that was shown to the user.

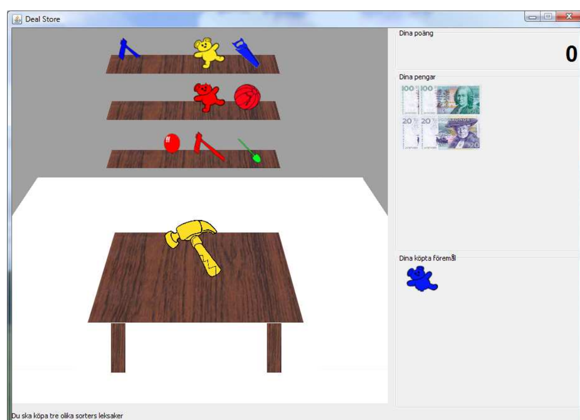

Figure 6: The user interface in DEAL. The object on the table is the one currently in focus. Example objects are shown on the shelf. Current game score, money and bought objects are shown on the right.

4.2 Speech segments in DEAL

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 standardized lexical expressions (cue phrases) or filled pauses. Such speech segments commit very little semantically to the rest of the utterance and are therefore very useful as initiations of utterances, since such speech segments can be produced immediately after the user has stopped speaking, allowing the Wizard to exploit the additional time to transcribe the rest of the utterance.

The DEAL corpus was used to create utterance 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 expression such as “ja”, “a”, “mm” (Eng: “yes”), filled pauses such as “eh”, “ehm” and expressions used to initiate different domain specific speech acts (for example “it costs” and “let me see”) were extracted. The segments were re-synthesized using Expros, a tool for experimentation with prosody in diphone voices (Gustafson & 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 generated text segments (for example references to objects, prices, etc) that were synthesized and generated on-line with the same diphone voice.

An example interaction with the incremental version of the system is shown in Table 3. S.11 exemplifies a self-correction, where the system prepares to present another bid, but then realizes 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.

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

Table 3: An example DEAL dialogue (translated from Swedish). Speech segments are marked in brackets.

4.3 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 a mission: to buy three items (with certain characteristics) in DEAL at the best possible price from the shop-keeper. 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 dialogue system and were not aware of the Wizard-of-Oz set up. Each participant interacted with the system four times, first two times with each version of the system, after which a questionnaire was completed. Then they interacted with the two versions again, after which they filled out a second questionnaire with the same questions. The order of the versions was balanced between subjects.

The mid-experiment questionnaire was used to collect the participants' first opinions of the two versions 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 messages in the system were logged.

4.4 Results

Figure 7 shows the difference in response time between the two versions. As expected, the incremental version started to speak more quickly (M=0.58s, SD=1.20) than the non-incremental version (M=2.84s, SD=1.17), while producing longer utterances. 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 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 may 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=5.02s, SD=1.54) is about 600ms faster than the average for non-incremental version (M=5.66s, SD=1.50), ($t(704)=5.56$, $p<0.001$).

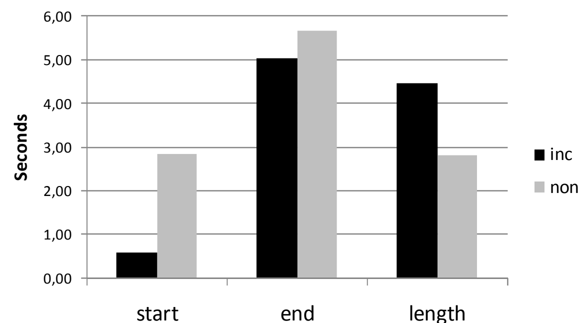

Figure 7: The first two column pairs show the average time from the end of the user's utterance to the *start* of the system's response, and from the end of the user's utterance to the *end* of the system's response. The third column pair shows the average total system utterance *length* (end minus start).

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 the difference, 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 Table 4. As the table 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.

	diff	z-value	p-value
preferred	0.23	-1.24	0.214
human-like	0.15	-0.76	0.445
polite	0.40	-2.19	0.028*
efficient	0.29	-2.08	0.038*
intelligent	0.11	-0.70	0.484
faster response	0.26	-1.66	0.097
feedback	0.08	-0.84	0.400
when to speak	0.35	-2.38	0.017*

Table 4: The results from the second questionnaire. All differences are positive, meaning that they are in favour of the incremental version.

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 (e.g., Bell, 2003). In order to examine whether the different versions affected the user's behaviour, we analyzed both the user utterance length and user response time, but found no significant differences between the interactions with the two versions.

5 Conclusions & Future work

This paper has presented a first step towards incremental speech generation in dialogue systems. The results are promising: when there are delays in the processing of the dialogue, it is possible to incrementally produce utterances that make the interaction more efficient and pleasant for the user.

As this is a first step, there are several ways to improve the model. First, the edges in the `SpeechPlan` could have probabilities, to guide the path planning. Second, when the user has finished speaking, it should (in some cases) be possible to anticipate how long it will take until the processing is completed and thereby choose a more optimal path (by taking the length of the `SpeechSegment`'s into consideration). Third, a lot of work could be done on the dynamic generation of `SpeechSegment`'s, considering syntactic and pragmatic constraints, although this would require a speech synthesizer that was better at convincingly produce conversational speech.

The experiment also shows that it is possible to achieve fast turn-taking and convincing responses in a Wizard-of-Oz setting. We think that this opens up new possibilities for the Wizard-of-Oz paradigm, and thereby for practical development of dialogue systems in general.

6 Acknowledgements

This research was funded by the Swedish research council project GENDIAL (VR #2007-6431).

References

Bell, L. (2003). *Linguistic adaptations in spoken human-computer dialogues. Empirical studies of user behavior*. Doctoral dissertation, Department of Speech, Music and Hearing, KTH, Stockholm.

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*.

Fraser, N. M., & Gilbert, G. N. (1991). Simulating speech systems. *Computer Speech and Language, 5*(1), 81-99.

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)* (pp. 293-296). Berlin/Heidelberg: Springer.

Hjalmarsson, A., Wik, P., & Brusk, J. (2007). Dealing with DEAL: a dialogue system for conversation training. In *Proceedings of SigDial* (pp. 132-135). Antwerp, Belgium.

Hjalmarsson, A. (2008). Speaking without knowing what to say... or when to end. In *Proceedings of SIGDial 2008*. Columbus, Ohio, USA.

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*. Technical Report RR-95-11, German Research Center for Artificial Intelligence.

Levelt, W. J. M. (1989). *Speaking: From Intention to Articulation*. Cambridge, Mass., USA: MIT Press.

Raux, A., & Eskenazi, M. (2008). Optimizing endpointing thresholds using dialogue features in a spoken dialogue system. In *Proceedings of SIGdial 2008*. Columbus, OH, USA.

Schlangen, D., & Skantze, G. (2009). A general, abstract model of incremental dialogue processing. In *Proceedings of EACL-09*. Athens, Greece.

Skantze, G., & Schlangen, D. (2009). Incremental dialogue processing in a micro-domain. In *Proceedings of EACL-09*. Athens, Greece.