User-initiated Sub-dialogues in State-of-the-art Dialogue Systems

Staffan Larsson
Centre for Linguistic Theory and Studies in Probability (CLASP)
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
sl@ling.gu.se

Abstract

We test state of the art dialogue systems for their behaviour in response to user-initiated sub-dialogues, i.e. interactions where a system question is responded to with a question or request from the user, who thus initiates a sub-dialogue. We look at sub-dialogues both within a single app (where the sub-dialogue concerns another topic in the original domain) and across apps (where the sub-dialogue concerns a different domain). The overall conclusion of the tests is that none of the systems can be said to deal appropriately with user-initiated sub-dialogues.

Index Terms: dialogue, dialogue systems, dialogue management, human-machine interaction, dialogue structure

1 Introduction

This paper follows Larsson (2015) in taking a look at how dialogue systems from some of the major players on the market actually deal with some conversational behaviours frequently encountered in human-human dialogue. It should be noted that the tests necessarily reflect the behaviour of the systems tested at the time of the test. As any other app in your mobile, conversational agents are frequently updated and new behaviours are added. The tests described here were carried out in March 2017.

The work presented here builds on the "Trindi Tick-list" (Bos et al., 1999) which was constructed in the TRINDI project[1] to examine whether certain dialogue behaviours can be reliably manifested by a dialogue system. The original tick-list is still being used (Hofmann et al., 2014), and there have been later revisions and amendments (although these remain to be published). With the advent of widely available spoken dialogue systems in smartphones, the kind of evaluation exemplified by the Trindi Tick-list has again become relevant.

In this paper, we will choose a small subset of the questions in the current tick-list, and investigate how systems deal with dialogue behaviours related to *user-initiated sub-dialogues*, i.e. cases where a system question is responded to with a user question (or request). According to Łupkowski and Ginzburg (2013), responding to a query with a query is a common occurrence, representing on a rough estimate more than 20% of all responses to queries found in the British National Corpus. Also, many of us are used to being able to multi-task using our computers and smartphones, jumping back and forth at will between several apps or programs, and there seems to be no particular reason why we should not be able to do so just because we are interacting using spoken dialogue.

2 The systems in the test

We tested five systems: Siri[2], API.AI[3], Houndify[4], Cortana[5] and Alexa[6]. The choice of these systems was based on (1) availability, (2) being reasonably well-known, and (3) allowing testing the dialogue phenomena in question[7]. While previous tests (Larsson, 2015) used complete off-the-shelf

[1]http://www.ling.gu.se/projekt/trindi/

[2]http://www.apple.com/ios/siri/
[3]https://api.ai/
[4]https://www.houndify.com/
[5]https://www.microsoft.com/en-us/mobile/experiences/cortana/
[6]https://developer.amazon.com/alexa
[7]For example, the Google Assistant and Google Home systems rarely if ever ask questions to the user; instead, they generally try to take whatever information they have and do something with it. This means that there no natural place to initiate a sub-dialogue when interacting with these systems. For these reasons, they have not been included in this test.

end-to-end dialogue applications (e.g. for calling people up), the market has shifted towards offering developers various degrees of freedom and support in implementing dialogue applications on top of a dialogue system (or dialogue system platform). In this respect, the systems in the test differ to a large extent – not only with respect to the extent to which they support various dialogue behaviours, but also as to whether they offer any dialogue management capabilities at all. Roughly speaking, the systems fall into three broad classes:

- **Closed systems**: A fixed set of non-configurable dialogue applications (e.g. Siri).
- **Configurable service platforms** provide dialogue management and domain implementations[8]; developers select domains and connect services to these ready-made domain implementations (e.g. SiriKit, Houndify[9]).
- **Domain development platforms** provide generic dialogue management; developers implement their own domains or select from a set of predefined domains (e.g. API.AI, (Houndify[10]))
- **Dialogue shells** offer ASR, NLU and TTS; developers implement dialogue managers (including domain implementations) (e.g. Cortana, Alexa).

In Section 3, we discuss some complications arising from applying a single test to systems of all four classes. First, however, we provide a brief description of each of the tested systems.

2.1 Siri

Siri runs on the iPhone and on a variety of Apple devices. SiriKit[11] offers some minimal opportunities for developers to connect their own external services to Siri, but only for a limited range of service types (currently VoIP calling, messaging, payments, photo and workouts) for which ready-made language understanding and dialogue management knowledge is provided by Siri (and unavailable for developers). For each service type, a fixed set of "intents" (tasks) are defined, that the developer use to connect their service. In the current tests, we used ready-made Siri applications on an iPhone,

[8]Roughly, we use *(dialogue) app* to refer to the entity with which a user communicates about a certain domain, given some *domain implementation* encoding the knowledge required to talk about that domain.

[9]The openly available Houndify only allows accessing existing domains.

[10]Building custom domains for Houndify is currently by invitation only. We have not been able to test this feature.

[11]`https://developer.apple.com/sirikit/`

2.2 API.AI

API.AI (which can be used in Google Assistant and Google Home apps) offers an interactive GUI tool for building a dialogue application by giving sample user sentences and mapping these onto interpretations in terms of intents and entities. The user-defined app can be combined with a number of pre-defined apps (not editable). For the current test, we used a combination of one "home-made" application and a selection of predefined domains, since this gave us the opportunity to define a domain with several intents as well as intents with multiple parameters (necessary for performing all our tests). Specifically, a simple phone domain was implemented by the author using the API.AI developer GUI. The tests were conducted using the text interface on the API.AI developer website.

2.3 Houndify

Houndify is very similar to API.AI but we have so far not been able to get access to the developer tools. For this reason, we used only predefined applications in the tests. The tests were conducted using the text interface on the Houndify developer website.

2.4 Cortana

Cortana runs on a variety of Windows devices, and essentially allows developers to build apps that use Cortana's built-in ASR and TTS (as well as the phone touch-screen for graphical output and haptic input) with a Cortana look-and-feel. This means that NLU, dialogue management and NLG need to be implemented more or less from scratch by the developer. In this test, we used existing ready-made Cortana domains on a Nokia Lumia phone.

2.5 Alexa

Amazon's Alexa runs on the Amazon Echo, and is similar to Cortana, except it also offers generic and configurable NLU capabilities. For our tests, we used ready-made Alexa domains. While broadly classified as a "dialogue shell", Alexa does offer a general mechanism for switching between domains ("skills") that is relevant for our current concerns. The tests were conducted on an Amazon Echo.

3 Complications

Our main interest is to evaluate *general* (domain independent) dialogue management features, which may be problematic in some cases where it is not clear if a certain behaviour is implemented in a general dialogue manager, or if it is produced by a domain-specific dialogue management script. In many cases, the source of an observed behaviour can be inferred from documentation, but in other cases more indirect evidence has to be used. For example, if a system displays identical behaviours across several domains, this may be evidence that it is produced by a general dialogue manager.

Note that we are not mainly interested in what is *possible* in a given system, but rather in what is *supported* by the system. That is, the developer should not have to implement all or most of the code required to deal with the dialogue feature in question. Ideally, the developer should not have to do anything to enable it (other than possibly selecting or deselecting the feature). In the case of "dialogue shells", very few dialogue features are supported. Pretty much any behaviour can *in principle* be implemented, but this is not necessarily very helpful for the developer.

Another problem concerns the notion of a domain (or "app"). Whereas in some cases it is clear whether two tasks (or "skills") are implemented as separate domains. We have assumed e.g. that asking about missed calls and calling people up both belong to the "phone" domain, while asking for the time or setting an alarm probably instead belong to the "clock" domain.

Despite these complications, we believe the tests in this paper can be of interest, and we have tried to make clear the specific characteristics of the systems to enable the reader to assess the reliability of the tests.

4 Results

The overall results of the tests are shown in Table 1. In this section, we present the details of these results and provide concrete interactions as evidence.

4.1 User anytime jump to task within app

The first three tests consider the capabilities of the systems to deal with user utterances that respond to a system question related to a task T with a request or question related to another task T' (where T and T' are both in the same domain). We refer to such a request or question as "jump to task". The "anytime" refers to the fact the the jump to T' can occur at any point during the dialogue related to task T (not only e.g. after T' is done). The tests are the following (indexed Fn for "feature n"):

- (F1) User anytime jump to task within app. Does the system respond adequately to jumps, i.e. does it shift the topic of conversation to T'?
- (F2) System resume after within-app subdialogue. After finishing T', does the system return the dialogue to the previous (unfinished) topic T?
- (F3) Signal task resumption. If the system resumes T, does it also indicate this somehow?

Note that F2 and F3 are not applicable if F1 is answered negatively, and that F3 is not applicable if F2 is answered negatively. Here is an example of a dialogue with a system that would yield "yes"-answers to all three questions:

U:	Make a call
S:	Who do you want to call?
U:	Do I have any missed calls?
S:	No. Returning to making a call. Who do you want to call?

Here, the system's last utterance (starting with "No") first shows that it can deal with the embedded question by answering it (F1). It also shows that it returns to the unfinished topic of making a call, by repeating the question "who do you want to call." (F2) Finally, it signals task resumption explicitly ("Returning to making a call") (F3).

Siri accepts the embedded question T' and responds to it, but does not manage to return to the previous unfinished topic T.

U:	Make a call
S:	Who do you want to call?
U:	Do I have any missed calls?
S:	No
U:	Okay
S:	-

	Siri	API.AI	Houndify	Cortana	Alexa
F1. User anytime jump to task within app	+	+	?	-	+
F2. System resume after within-app sub-dialogue	-	-	?	-	-
F3. Signal task resumption	N/A	N/A	?	-	N/A
F4. User anytime jump across apps	+	-	-	-	(+)
F5. System resume after other-app subdialog	-	N/A	N/A	N/A	-
F6. Signal app (and task) resumption	N/A	N/A	N/A	N/A	N/A

Table 1: Dialogue behaviours related to user-initiated sub-dialogues in five dialogue systems

API.AI produces the same behaviour as Siri, failing to return to the unfinished topic.

Unfortunately, **Houndify** could not be tested for this feature, since none of the available domains featured (1) more than one task, whereof (2) at least one task involved asking a question to the user.

Since **Alexa** does not implement calling people up, we instead used the clock domain for testing F1-F3. The results were the same as for Siri and API.AI.

U: Set the alarm
S: Alarm for what time?
U: What time is it?
S: 10.15
U: OK
S: -

Cortana did slightly worse than the other systems, failing to understand the embedded question (again we used the clock domain)[12].

U: Set the alarm
S: Set you alarm for what time?
U: What time is it?
S: What time did you want the alarm?

4.2 User anytime jump across apps

The next three tests (F4-F6) consider the capabilities of the systems to deal with user utterances that respond to a system question related to a task T in a domain D with a request or question related to a task T' in another domain $D' \neq D$. We refer to such a request or question as ”jump across apps”. Again, ”anytime” refers to the fact the the jump to T' can occur at any point during the dialogue related to task T (not only e.g. after T' is done). The tests are the following:

- (F4) User anytime implicit jump across apps. After having asked a question related to a task T in a domain D, does the system respond adequately to a request or question related to a task T' belonging to a domain D' (but not mentioning D'), i.e. does it shift the topic of conversation to T' and D'?
- (F5) System resume after other-app sub-dialogue. After finishing T', does the system return the dialogue to the previous (unfinished) topic T?
- (F6) Signal app (and task) resumption. If the system resumes T, does it also indicate this somehow?

Note that F5 and F6 are not applicable if F4 is answered negatively, and that F6 is not applicable if F5 is answered negatively. Here is an example of a dialogue with a system that would yield ”yes”-answers to F4, F5 and F6:

U: Make a call
S: Who do you want to call?
U: What time is it?
S: 8 pm. Returning to making a call. Who do you want to call?

Siri offers user anytime jump across apps[13]. The behaviour concerning F5 and F6 parallels that of F2 and F3.

U: Call John
S: Which number for John?
U: What time is it?
S: 8 pm.
U: OK
S: -

API.AI does not provide F4:

[12]Cortana had no problem understanding ”What time is it” when the question was not embedded.

[13]This assumes that calling people up and asking the time belong to different domains

U: Make a call
S: Who do you want to call?
U: What time is it?
S: Who do you want to call?

For **Houndify**, we used a different combination of domains to test for F4-F6, with tip calculation taking the place of calling someone up. The result is similar to that for API.AI, but instead of simply ignoring the embedded request, the system indicates that it does not understand it.

U: Calculate tip
S: What is the amount of the bill?
U: What time is it
S: Sorry, I did not understand that. The bill amount is indicated on the check. What is the amount of the bill?

The results for **Cortana** were similar to those for API.AI.

In many cases, **Alexa** requires the user to explicitly address the app, thus requiring what we may call ”gated commands” as opposed to ”gateless commands” (wich can be issued without explicitly addressing the relevant app).

U: Alexa, play Abba *on Spotify*

Regardless of whether gated or gateless commands were used, Alexa would switch topic as requested, but did not manage to return to the original topic.

U: Alexa, set the alarm
S: Alarm for what time?
U: Alexa, play Abba on Spotify
S: (plays music)

5 Discussion and future work

An obvious extension of the work presented here is to include more systems, i.e. Luis (from Microsoft) and Watson (from IBM). This also points to the need for regularly testing both new and established systems for a wide range of dialogue phenomena, preferably in a standardised manner.

Another obvious extension of the work presented here would be to relate the various dialogue behaviours to measurements of the quality, usefulness and attractiveness of dialogue systems that have or lack the respective features. Here, the PARADISE framework (Walker et al., 1997) could potentially be very useful. Such investigations, however, must take into account variability in the usefulness of various dialogue features with respect to the overall activity and other situational factors. A feature which is very useful in one context may be of little interest in another.

It seems likely that at least in some cases, the user may not expect or want a conversational partner to return to a previous topic. For example, the user may switch to another topic as a way of steering the conversation away from the current topic. How to distinguish cases where a user initiative is intended an interruption of the ongoing topic, vs. when it is intended as an embedded subdialogue, is an interesting area for future research.

It is also possible that real-time factors may play a role. If embedded sub-dialogues can be dealt with in an efficient and highly interactive manner, with minimal delay between turns, this reduces the user’s perceived cost (in terms of time and effort) of entering into a sub-dialogue, and may boost the usefulness of such sub-dialogues.

It should be noted that although none of the tested systems dealt adequately with user-initiated sub-dialogues, there are systems that do handle these phenomena. We know of at least two such systems, Indigo[14] from Artificial Solutions, and the Talkamatic Dialogue Manager (TDM) from Talkamatic[15,16]. These systems deal appropriately with most of the phenomena listed in Table 1[17].

6 Conclusion

We have tested five different well-publicised dialogue systems for their behaviour in response to user-initiated sub-dialogues within and across apps. The overall conclusion of the tests is that none of the systems tested deal appropriately with user-initiated sub-dialogues. In light of how frequent this behaviour is in human-human dialogue, we regard this as a serious shortcoming.

We hope that the kind of evaluation presented here can improve our understanding of the state of the art in commercial dialogue systems, and suggest ways in which to improve such systems with respect to dialogue management.

[14] `http://www.hello-indigo.com/`

[15] `talkamatic.se`

[16] For transparency, it should be noted that the author is co-founder and co-owner of Talkamatic AB.

[17] TDM handles all of F1-F7. Indigo handles F1-F4 and F6-F7. However, Indigo has trouble with the over- and other-answering tests described in Larsson (2015).

References

Johan Bos, Staffan Larsson, I Lewin, C Matheson, and D Milward. 1999. Survey of existing interactive systems. Technical Report D1.3, TRINDI (Task Oriented Instructional Dialogue) project.

Hansjörg Hofmann, Anna Silberstein, Ute Ehrlich, André Berton, Christian Müller, and Angela Mahr. 2014. Development of speech-based in-car hmi concepts for information exchange internet apps. In *Natural Interaction with Robots, Knowbots and Smartphones*, Springer, pages 15–28.

Staffan Larsson. 2015. The state of the art in dealing with user answers. In Christine Howes and Staffan Larsson, editors, *Proceedings of the 19th Workshop on the Semantics and Pragmatics of Dialogue (goDIAL)*.

Paweł Łupkowski and Jonathan Ginzburg. 2013. A corpus-based taxonomy of question responses. In *IWCS 2013 (International Workshop on Computational Semantics)*.

M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *Proc. of the ACL*. Madrid, pages 271–280.