INPROTK IN ACTION: OPEN-SOURCE SOFTWARE FOR BUILDING GERMAN-SPEAKING INCREMENTAL SPOKEN DIALOGUE SYSTEMS

Timo Baumann, Okko Buß and David Schlangen

Department of Linguistics, Potsdam University
{timo|okko|das}@ling.uni-potsdam.de

Abstract: We present INPROTK, a toolkit for building incremental spoken dialogue systems. *Incremental* spoken dialogue systems (systems that may react while the user's utterance is ongoing) are a fairly recent research topic and allow for exciting new features. Even though toolkits exist that help in building conventional dialogue systems, INPROTK offers both a tested architecture for building incremental SDSs as well as many of the building blocks necessary when building such systems. With INPROTK a researcher can avoid many of the technical difficulties, which hopefully further fosters research in this area.

1 Introduction

Building a Spoken Dialogue System (SDS) is a complex task, as it requires a wide range of knowledge spanning areas such as dialogue and task management, interaction design, but also (often technical) skills in adjacent areas such as speech recognition, speech synthesis, and software engineering. Even though a researcher may be a specialist in one of the areas involved, it is hard to cover all aspects in full depth. Research in SDSs necessarily involves building such sytems, because end-to-end evaluations remain the most convincing way to determine whether improvements to a component scale to the application level: High-level metrics like transaction success and completion time can only directly be measured in a complete system. As a consequence, dialogue researchers spend a lot of time with implementation details that are not in the main line of their research interest.

To facilitate research especially in the area of *incremental* spoken dialogue systems, we present INPROTK, a toolkit for building such systems with a focus on *incrementality*, i. e. the processing of minimal amounts of input as soon as they become available. This new research trend is in contrast to the "conventional" way of dialogue processing, in which incrementality was limited to a speaker's *turns*, with complete turns being processed and responses generated only once a turn is over. Truly incremental SDSs allow many new features such as immediate feedback, back-channel generation and faster turn-taking, resulting in a leap forward in SDS design. Recent research shows that incremental SDSs are preferred over their non-incremental counterparts (e. g. [1]). INPROTK offers solutions to many of the problems one faces when building a research SDS with a focus on incrementality, while offering a flexible, and easily extensible modular architecture.

Our toolkit implements the general model of incremental dialogue processing outlined in [15], resulting in a system that is incremental from the ground, while offering fine-grained control over the level of incrementality. As our system is tightly integrated with a speech recognition toolkit (Sphinx-4, [17]), we are able to provide "rich" speech recognition results with phonetic and prosodic information easily accessible for all modules in the processing pipeline of the

dialogue system. Finally, the toolkit is being developed and constantly tested with German language SDSs, which make it especially valuable for the German research community. (But note that INPROTK is in no way limited to German.) Our toolkit is written in the Java programming language, and is available as open source software.[1]

The remainder of this paper is structured as follows: In Section 2 we briefly review other toolkits for SDSs, before detailing our architecture in Section 3. We give an overview of the generic incremental modules in our toolkit in Section 4 and show example systems built with the toolkit in Section 5. We conclude with a discussion of issues for future work in Section 6.

2 Related Work

Toolkits for developing end-to-end dialogue systems certainly exist. Commercial vendors (Voxeo, IBM, and others) provide a programmable, often graphic, layer on top of industry standards like VoiceXML[2]. Such systems are, however, generally unsuitable for research applications for a number of reasons: license fees; limited or no access to source code; focus on industry standards, which, while useful for standard application development, are too narrowly defined for exploring larger dialogue issues that are of research interest. Most importantly, none are capable of implementing fully incremental dialogue systems.

In the research world, the CMU Communicator Toolkit [14] and its successor Olympus/RavenClaw [4] are architectures that allow for a modular design of SDSs, with modules Modules connected in a strict pipeline. This approach has yielded a range of research applications (e.g. AT&T [10], Let's Go! [13]). Even though this toolkit has been shown to work with highly reactive systems with fast turn-taking capacities [12], it has not been used (or shown to be useful) for building fully incremental ones.

TrindiKit [9] and Dipper [5], represent toolkits with dialogue managers that follow the information-state-update approach (ISU) [9]. These toolkits also provide modular architectures, however the interface between input and output speech resources implement a specific ISU mechanism. Not all modules, especially dialogue management ones, are therefore as easily replaced, making it difficult to change components that follow a different paradigm. The ISU approach, while easily adapted to allow incremental processing in theory [7], is not implemented with previsions for incrementality in these toolkits, possibly requiring substantial changes.

3 Architecture

The architecture of INPROTK is based on the abstract model of incremental dialogue processing outlined in [15]. In this model, the dialogue system consists of a (static) network of processing *modules*. Each module has a *left buffer*, a *processor*, and a *right buffer*, where the normal mode is to take input from the left buffer, process it, and provide output in the right buffer, from where it goes to the next module's left buffer.

Modules exchange *incremental units* (IUs), which are the smallest 'chunks' of information that can trigger connected modules into action. IUs typically are part of larger units; e. g., individual words as parts of an utterance. This relation of being part of the same larger unit is recorded through *same level* links; the information that was used in creating a given IU is linked to it via *grounded in* links. As IUs are interconnected with these links, a network representing all the information known to the system emerges. This network is highly dynamic, with changes to the network reflecting the system's changing internal state over time.

[1] INPROTK is available at `http://www.ling.uni-potsdam.de/~timo/code/inprotk/`.

[2] `http://www.w3.org/TR/voicexml21/`

Modules have to be able to react to three basic situations: that IUs are *added* to the network (and consequently the module's buffer), which triggers processing; that IUs that were erroneously hypothesised 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 (or, respectively, expect it to not be revoked anymore). Our implementation of [15] is deliberately limited to event-based, bottom-up processing (with no direct support of expectation-based top-down processing) which greatly reduces the complexity of inter-module communication.

3.1 Incremental Units

Incremental Units (IUs) are typed objects, where the base class `IU` specifies the links (same-level, grounded-in) that allow to create the IU network, handles the assignment of unique IDs, and some other basic operations. A design principle with the IU network is to make all relevant information available, while avoiding replication: For instance, an IU holding a bit of semantic representation can query which interval of input data it is based on, where this information is retrieved from the appropriate IUs by automatically following grounded-in links.

The payload and specific properties of an IU are specified in its *type*. There are several predefined subtypes of IUs in the framework which represent different types of information and support different kinds of query operations. Most modules only expect input IUs of a specific type and deliver output IUs of another type or types. For example, `SegmentIU`s, `SyllableIU`s and `WordIU`s represent phonemes, syllables and words, respectively. As IUs are proper objects, they may provide complex operations, carrying out significant amounts of computations on their own. For example, a word can be queried for its accentuation status, and will in turn query features of its associated syllables and phonemes.

3.2 Modules and Inter-Module Communication

As mentioned above, module communication is event-based with information flowing from bottom (i. e. input audio) to top (i. e. dialogue management and further on to audio dispatching). However, the modules do not necessarily form a pipeline, as one module may output to and receive input from several other modules. The flow of information through the module network is (statically) configured in a system configuration file which specifies which modules *listen* to which. Obviously, most modules can only be connected to certain kinds of other modules.

Modules *push* information to their right whenever they have new output information, a new *state*, available. A hypothesis change is (redundantly) characterised by passing both the complete current buffer *state* (a list of IUs) as well as the *delta* between the previous and the current state. We have decided for this redundant output, to leave listening modules a broader choice of implementation: Some module may prefer to carry out its computations based on the complete preceding module's output, while another may prefer to only look at the most recent changes.

As currently implemented, all modules live in the same process. However, inter-module communication may transparently use a messaging protocol (we have previously used the OAA but other communication layers could also be used) by plugging pairs of appropriate communication modules into the module network. In such a use-case it may be profitable to only send deltas and to leave the reconstruction of the full state to the receiving side.

Modules can be fully event-driven, only triggered into action when notified, or they can run persistently which allows them to create endogenous events like time-outs. Our modules are currently not running persistently, but instead register time-outs with a a separate component and are notified if the time-out runs out, as outlined in the next subsection.

3.3 Signals

We believe that some events are not correctly modelled as incremental hypothesis changes (i. e. within the IU framework) in an incremental SDS. Such events include turn-taking events, initiation and tear-down of a dialogue in a telephony application, and the like. Additionally, in the event-driven approach, a module may not directly be notified that *no event* has happened for a while. Still, this information is crucial for SDSs in order to properly handle user time-outs (e. g. initiating a re-prompt). We provide a separate communication track along which *signals*, which are any kind of information that is not seen as incremental hypotheses about a larger whole but as information about a single current event, can be passed between modules. This communication track also follows the observer model, where processors define interfaces that listeners can implement and subscribe to corresponding events.

4 Predefined Modules

INPROTK comes with a number of generic and configurable modules for building incremental SDSs. The focus here is on input modules, as well as monitoring and output components. We also provide an incremental dialogue management (DM) module following the ISU approach. Notice however that research in this area has only just started which means that the incremental DM component is likely to change significantly in the future. Also, to try other dialogue management paradigms, one may have to implement new variants of this module.

4.1 Speech Recognition

Our speech recognition module is based on the Sphinx-4 [17] toolkit and comes with acoustic models for German.[3] The module queries the ASR's current best hypothesis after each frame of audio and changes its output accordingly, adding or revoking `WordIU`s and notifying its listeners. Additionally, for each of the WordIUs, `SyllableIU`s and `SegmentIU`s are created and bound to the word (and to the syllable respectively) via the grounded-in hierarchy. Later modules in the pipeline are thus able to use this lower-level information (e. g. to disambiguate meaning based on prosodic aspects of words).

To allow for *prosodic processing*, we inject additional processors into Sphinx' acoustic frontend which provide features for further prosodic processing (pitch, loudness, and spectral tilt). In this way, IUs are able to access the precise acoustic data (in raw and processed forms).

An ASR's current best hypothesis frequently changes during the recognition process with the majority of the changes not improving the result. Every such change triggers all listening modules (and possibly their listeners), resulting in a lot of unnecessary processing. Furthermore, changes may actually deteriorate results, if a 'good' hypothesis is intermittently changed for worse. Therefore, we developed *hypothesis smoothing* approaches [3] which greatly reduce spurious edits in the output at the cost of some timeliness: With a lag of 320 ms we reduced the amount of spurious edits to 10 % from an initial 90 % [3]. The current implementation of hypothesis smoothing is taylored specifically towards ASR output, but other input modules (like gesture or facial expression recognition) could easily be smoothed with similar methods.

Sphinx-4 can be run in *various search modes* which has consequences for the internal data ordering in the token-pass algorithm employed. Our speech recognition module covers all the different data orderings, enabling recognition against statistical language models and EBNF

[3] Models for English, French and other languages are available from the Sphinx' distribution and from `http://www.voxforge.org`.

grammars, as well as forced alignments. Audio may be input from the microphone, from a file (for testing purposes), and from the network (using RTP). Finally, in addition to ASR, there is also a text-based input module that also outputs `WordIU`s. This greatly facilitates the systematic debugging of later modules in the dialogue system.

4.2 Natural Language Understanding

A unification based semantic resolver takes care of the task of generating semantic representations for given word hypotheses. We represent semantics as matrices (AVMs) of attribute-value pairs (AVPs) which contain pertinent information about semantic entities, usually objects in the application domain. For example, in our puzzle domain with tiles to manipulate and fields in which to place them, a semantic entity TILE might be represented as shown in Example 1. Note that the value of LOCATION is in turn a (sub-)AVM.

(1) "*looks like a red cross in the top left corner*" TILE: [COLOUR: red; NAME: x; LOCATION: [ROW: top, COL: left]]

(2) "*a field coloured blue in the bottom right*" FIELD: [COLOUR: blue; LOCATION: [ROW: bottom, COL: right]]

The resolver works in two steps: (a) composition and (b) resolution of semantic representations. The underlying algorithm in both is unification. In our toolkit, each word may be associated with one or more *denotations* (in terms of AVPs) using a lexicon. Furthermore, a configuration file describes the *structure* of the AVMs allowed in the domain (in our example: TILEs may have names, LOCATIONs may consist of rows/columns, ...), and another configuration (from file or online) determines the currently *available objects* and concepts in the domain (e. g. the AVMs in Examples 1 and 2). We will now briefly outline the two resolution steps.

Composition produces underspecified AVMs based on incoming `WordIU`s. Starting from completely unspecified structures (as in the examples, but without any values filled in), a new word's AVP is unified to the set of AVMs. (This succeeds if an AVM contains an equal or underspecified and matching AVP.) In this manner, as more words come in, the AVM 'grows' more and more specific. Note that some AVPs may apply to more than one type of AVM. (In the example, COLOUR is part of both FIELD and TILE). For this case, the composer maintains two mechanism: composition of parallel hypotheses (i. e. all AVMs are filled with the value) and chunk marking. *Chunk marking* narrows the choice of target AVMs by interpreting user input according to given chunking rules. A chunk mark is defined as a word (or word sequence), which rules out certain AVMs. For instance, the word "*tile*" represents the beginning of a TILE chunk; likewise, a multi-word sequence such as "*into the*" signals that any ensuing AVPs belong to a FIELD chunk (as in "*move the tile into the blue field*").

The composed AVMs are then *resolved* with the list of fully specified AVMs for entities in the domain. This step succeeds if there exists an AVM in the domain for which all AVPs of a composed AVM have a corresponding AVP with matching attribute and value. The algorithm is recursively applied to sub-AVMs. The set of fully specified AVMs matching the (possibly underspecified) user input is passed on to the next module in a `SemIU`. Eventually a *singleton set* is the desired output of the semantic resolver, indicating that user input arrived at an unambiguous semantic representation (e. g. a specific tile or field in our game). If however multiple AVMs resolve when the utterance is over, these have to be disambiguated with the user by a 'later' module, such as the dialogue manager.

4.3 Floor Tracking

While the topic of turn-taking is most often solved trivially in conventional SDSs (i. e. a user's turn is over when the user stops speaking), turn-taking can become much more complex in a fully incremental system, when overlaps between turns, back-channels and the like are taken into account. In order to reduce the complexity of the dialogue manager proper, INPROTK offers a separate module to track whether the user requests the floor (i. e. the right to speak). The floor tracker tries to guess whether a user's silence is a signal to release the floor or whether she wants to *hold* it based on prosodic means and spoken words (e. g. rising pitch or pragmatic incompleteness are signs that the turn is not yet over). The floor tracker uses the signalling method outlined in Subsection 3.3 and the dialogue manager below consumes the signals.

4.4 Dialogue Management

Traditionally, a DM's role is to compute system output (speech or visual) based on hypotheses about user input. In incremental SDSs, we want to be able to do so while user input is ongoing. We have chosen a *questions under discussion* (QUD, [8]) based ISU approach to dialogue management similar to [9], but fully incremental. The DM consists of an *information state* (IS) and *update rules* that manipulate it. It interfaces with NLU, floor tracking and output components to update its state and to generate output.

The IS consist of a stack of 'questions', which in turn contain structures of semantic slots that have three fields: the slot's semantic value, an action associated with it as well as its grounding state. The semantic value of a slot is represented by an AVP which corresponds to the payload of a `SemIU`. Actions are on-screen or linguistic acts that the DM may execute when the rule fires. For instance, `prepare(A)` tells output producing modules to 'prepare' for performing action `A` by, say, changing cursor shape or moving the cursor. The *grounding state* describes the DM's belief about the action's status, which is relevant for the update rules described next.

Update rules consist of preconditions (PRE) and effects (EFF). The former are checks on the current IS, while the latter describe modifications on the IS to be performed if PRE holds. As a prose example, a rule's PRE conditions might express "if relevant semantic information becomes known", corresponding to EFF "fill the slot, queue its action for performance and set its grounding state to 'private'." After that action is produced a second rule might trigger whose PRE and EFF state "if an action is performed", "set its grounding state to public" respectively.

With these IS mechanisms, the DM can treat information from user input and produce as well as track its own output incrementally. For worked examples with more details about the grounding states, update rules and floor signals see [7] and [6].

4.5 Output and Monitoring

INPROTK provides an audio dispatch component which outputs (a) canned speech from files and (b) can be configured to call the Mary TTS server [16] for free-text synthesis. Output is sent to the speakers or via RTP over the network. In an incremental system it is vital to know which parts of the system's utterance have been output so far (e. g. if the system is interrupted during an enumeration). For this reason, the output component generates `WordIU`s during playback to which the dialogue manager (or another module) may be configured to listen to.

Additionally, there are several monitoring modules which can be plugged into the system just like (and in addition to) other modules, displaying a module's current hypothesis, dumping detailed representations of the IU network, or sending messages to TEDview [11], a viewer for incrementally changing linguistic data.

5 Example Systems

In this section we demonstrate two example systems built with INPROTK. As the toolkit uses a highly modular architecture, systems can exchange or extend modules with proprietary solutions as needed. While the first system below uses all the off-the-shell modules described above (with a few extensions for graphical output) in a moderately difficult naming task, the second system implements a task-specific NLU component and a different approach to action management than outlined above in a speech-controlled game-like application, underlining the flexibility and extensibility of INPROTK.

5.1 A Multi-Modal Puzzle Game

Our prototype that uses the NLU and DM modules described above allows a user to name and manipulate tiles from a puzzle domain. The main task this system performs is to select or delete tiles on a canvas display according to user input. Users can use a number of referring expressions to refer to the tiles on display (by colour, by shape, or name) and the desired action. During user input, the system displays and grounds understanding on the game canvas. Example 3 shows the various modules' outputs as a sample utterance unfolds. Notice how partial understanding already leads to actions being carried out and the interplay between floor tracker and dialogue manager.

(3)

ASR	*"the*	*red*	*cross*	*(silence)*	*delete*	*it"*	*(silence)*
FT	`start`			`BC cue`			`turn end`
NLU		TILE:t_1|t_2|t_4	TILE:t_1		ACT:delete		
DM			`highlight`(t_1)	`prompt(BC)`	`prepare(del)`	`del`(t_1)	`prompt(CONFIRM)`
OUT			cursor to t_1	"*yes?*"	cursor changes	t_1 deleted	"*was that right?*"

5.2 Highly Interactive Speech Control

We used INPROTK to control a simulated robot arm using speech input. In this domain (first outlined in [2]), reacting during the user's speech (which is only possible for incremental systems) is vital. Using the incrementality features of INPROTK, this system allows the precise positioning of the robot arm (e. g. saying "*stop*" almost immediately stops the robot arm).

For the system, we built a custom NLU module, which turns the sequence of words directly into what we call `ActionIU`s. `ActionIU`s themselves contain the logic necessary to execute their associated actions, or to revert an action in case the underlying words are changed by the ASR by using the information available in the IU network using same-level and grounded-in links. The IU framework is specifically suited for this very-low latency system, as some actions may have to be revoked (when ASR hypotheses change), which is easy for the `ActionIU`s to carry out via the same-level links. In effect, this obsoletes the concept of a dialogue or action management processor and moves further processing into the IU network.

6 Discussion

We have introduced INPROTK, a toolkit for building incremental SDSs. We have explained its modular architecture, which is based on the notion of *incremental units*, which are generated and processed by *incremental modules*, and via links form a network that represents all available information in the dialogue. We have presented the modules that come with INPROTK and which can be used to build SDSs that are incremental from the ground up. Finally, we have demonstrated two example applications which are powered by the toolkit.

Incremental speech processing is a young research area with many directions that are still unexplored. INPROTK offers tools to further such investigations. In our own future work, we will improve its treatment of dialogue management, floor tracking, and action management (i. e. deciding when –during an utterance?– and how –via what modality?– to dispatch a response). We hope that it will be of use to others as well, and that they will take it in further, yet unexplored directions.

Acknowledgements This work was funded by a DFG grant in the Emmy Noether programme.

References

[1] AIST, G., J. ALLEN, E. CAMPANA, C. GALLO, S. STONESS, M. SWIFT and M. TANENHAUS: *Incremental Dialogue System Faster than and Preferred to its Nonincremental Counterpart*. In *Proc. of the 29th Conference of the Cognitive Science Society*, Nashville, USA, 2007.

[2] BAUMANN, T.: *Integrating Prosodic Modelling with Incremental Speech Recognition*. In *Proceedings of DiaHolmia (Semdial 2009)*, Stockholm, Sweden, 2009.

[3] BAUMANN, T., M. ATTERER and D. SCHLANGEN: *Assessing and Improving the Performance of Speech Recognition for Incremental Systems*. In *Proceedings of NAACL-HLT*, Boulder, USA, 2009.

[4] BOHUS, D. and A. I. RUDNICKY: *The Ravenclaw Dialog Management Framework: Architecture and Systems*. Computer Speech & Language, 3(23):332–361, July 2009.

[5] BOS, J., E. KLEIN, O. LEMON and T. OKA: *DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture*. In *Proc. of SigDial*, Sapporo, Japan, 2003.

[6] BUSS, O., T. BAUMANN and D. SCHLANGEN: *Collaborating on Utterances with a Spoken Dialogue System Using an ISU-based Approach to Incremental Dialogue Management*. In *Proceedings of SigDial*, Tokyo, Japan, 2010.

[7] BUSS, O. and D. SCHLANGEN: *Modelling Sub-Utterance Phenomena in Spoken Dialogue Systems*. In *Proceedings of SemDial*, Poznan, Poland, 2010.

[8] GINZBURG, J.: *Interrogatives: Questions, Facts and Dialogue*. In *The Handbook of Contemporary Semantic Theory*. 1996.

[9] LARSSON, S. and D. TRAUM: *Information State and Dialogue Management in the Trindi Dialogue Move Engine Toolkit*. Natural Language Engineering, 6(3&4):323–340, 2000.

[10] LEVIN, E., S. NARAYANAN, R. PIERACCINI, K. BIATOV, E. BOCCHIERI, G. D. FABBRIZIO, W. ECKERT, S. LEE, A. POKROVSKY, M. RAHIM, P. RUSCITTI and M. WALKER: *The AT&T-DARPA communicator mixed-initiative spoken dialog system*. In *Proc. of ICSLP*, Beijing, China, 2000.

[11] MALSBURG, T. V. D., T. BAUMANN and D. SCHLANGEN: *TELIDA: A Package for Manipulation and Visualisation of Timed Linguistic Data*. In *Proceedings of SigDial*, London, UK, 2009.

[12] RAUX, A. and M. ESKENAZI: *A Finite-State Turn-Taking Model for Spoken Dialog Systems*. In *Proceedings of NAACL*, Boulder, Colorada, 2009.

[13] RAUX, A., B. LANGNER, A. BLACK and M. ESKENAZI: *LET'S GO: Improving Spoken Dialog Systems for the Elderly and Non-natives*. In *Proc. of Eurospeech*, Geneva, Switzerland, 2003.

[14] RUDNICKY, A., E. THAYER, P. CONSTANTINIDES, C. TCHOU, R. STERN, K. LENZO, W. XU and A. OH: *Creating Natural Dialogs in the Carnegie Mellon Communicator System*. In *Proc. of Eurospeech*, Budapest, Hungary, 1999.

[15] SCHLANGEN, D. and G. SKANTZE: *A General, Abstract Model of Incremental Dialogue Processing*. In *Proceedings of EACL*, 2009.

[16] SCHRÖDER, M. and J. TROUVAIN: *The German text-to-speech synthesis system MARY: A tool for research, development and teaching*. Int. Journal of Speech Technology, 6(4):365–377, 2003.

[17] WALKER, W., P. LAMERE, P. KWOK, B. RAJ, R. SINGH, E. GOUVEA, P. WOLF and J. WOELFEL: *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*. Techn. Rep., Sun Microsystems Inc., 2004.