Designing The Interaction For Pilot Decision Assistance: State Machines or Learning From Story Examples?

Khait S., Ardila-Torres N., Bernard D., Gouëzec F., Litvova M.
AIRBUS S.A.S, Toulouse, France
soufiane.khait@airbus.com, natalia.ardila-torres@airbus.com, denys.bernard@airbus.com, frederic.gouezec@airbus.com, monika.litvova@airbus.com

INTRODUCTION

With the increasing complexity of the aeronautical environment, the cockpit designers will develop virtual assistance functions to support pilots in operational situations such as preparing a flight, managing a failure, or defining an approach strategy. In complex situations, decisions are co-elaborated by the pilot and the assistant. Adaptive interaction patterns must be implemented. This paper addresses some methodological aspects of the design of such adaptive interaction, in the specific context of cockpit development processes.

Many classical approaches for designing collaborative systems model user-system interaction as synchronized state machines. (Winograd, 1986), (Bui, 2006), (Goronzy, 2006), (Degani, 2013), (Zamanirad, 2020). But recent chatbot design frameworks are promoting alternative methods for specifying and implementing user-system interaction (Metalinou, 2020) (Warwedam, 2020): a "*model*" of the interaction is obtained by training a neural network on a set of "*stories*". The model "*learns*" how to decide what next action is to be taken at each step of the interaction. This approach is deemed effortless and more robust to unexpected situations than classical approaches (Nicol, 2017). Our benchmark study compares both approaches from an industrial perspective. The basis for evaluation is a set of requirements tailored to the development of cockpit decision assistants: the method shall allow the designers to cover diverse and complex operation patterns, to take advantage of multimodal interaction in the cockpit, to co-design the system with pilots, and to demonstrate a target design assurance level. Although this is still an ongoing work, we would like to share the goals and methodology in this conference, because of the possible impact on the way Airbus will work with its partners on this subject. This abstract presents: the system development process applied to human-system interaction; the two considered modeling techniques; the test architecture put in place for the study; our requirements for the modeling framework; concluding discussion elements, including intermediate evaluation results.

INTERACTION DEVELOPMENT PROCESS

The variety of cockpit operational situations implies a large variety of pilot-assistant interaction patterns. To compare them and to assess their suitability, these steps executed:

- Operational analysis specifies human-system interaction as operational scenarios. This phase requires frequent workshops with pilots.
- Following the operational analysis, a functional analysis defines the functional breakdown of the system and requirements are cascaded to functions.
- Interaction requirements and operational scenarios are validated and verified through: discussions and workshops with pilots; tests on simple and scripted interaction examples, tests with functional prototypes or simulations; tests on representative simulators or integration benches.

- The implementation of a human-system interaction is usually performed by equipment/component providers. A final verification takes place after equipment delivery. This verification activity is preferably done on integration benches.

A generic functional architecture pattern is applied for decision assistance functions. The behavior of the virtual assistant is driven by a "dialog manager", which decides the next dialog act to be taken by the assistant at each step of the dialogue. Its inputs are mainly "dialog acts" coming from an "interpreter" which interprets pilots action as intentional dialog acts (e.g. pushing a button, saying or writing something, clicking somewhere on a touchscreen). The dialog acts taken by the assistant are manifested by a dedicated function by using the adequate modality. The goal of our benchmark is to compare several ways of specifying, implementing and testing the dialog manager.

CANDIDATE APPROACHES FOR DIALOGUE MANAGEMENT

Approach 1: Dialog manager as a hierarchical state machine

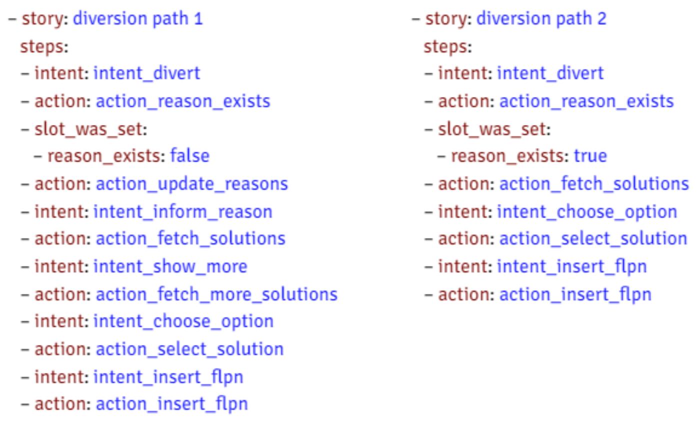
Modeling dialog or protocols via state machines has been a widespread practice for a long time. A usual choice for conversation modeling is to assign a state diagram to each interlocutor. Transitions in those state diagrams are triggered by dialog acts taken by the other participant (Winograd, 1986). In this study, we use Hierarchical State Machines (HSMs), which are finite state machines whose states themselves can be other state machines. They are a well founded, standardized formalism supported by industrial toolboxes and standards like UML (Yannakakis, 2000). In the case of the pilot's multi-skill assistant, we represent the state of the overall assistant as the combined (parallel) state of each of the skills. The events which trigger the transitions are dialog acts from the pilot, and will change the state of any skill whose state machine is waiting for this type of dialog act.

Approach 2: Learning dialogue from stories examples

Other alternatives for managing dialogue rely on probabilistic models. Two such approaches can be distinguished: Markovian models and Machine Learning (ML) based models. Among the Markovian models, we can mention the Partially Observable Markov Decision Process (POMDP) models (Roy, 2000). Such models are used for planning under uncertainty, which gives the agent the ability to estimate the outcome of its actions even when it cannot exactly observe the state of its environment. Among the machine learning based models, several deep learning models have been studied, such as Recurrent Neural Networks (RNN) architectures (Henderson et al, 2013) and other RNN types, including Long Short Term Memory (LSTM) (Williams, 2015). The idea is to select, at each new utterance submitted by the user, the action which has the best probability computed over a finite action domain.

More recently, Transformers based models (Vlasov, 2020) have shown promising results in dialogue management. The self-attention of this architecture makes it possible to ignore parts of the dialogue that are unrelated to the present expected task. For our study, we selected the RASA Open Source framework (RASA Open Source, 2021) which implements a Transformers based policy. Here, the model of the dialog manager (DM) has to be trained upon a set of stories which are example dialogues between a human and a virtual assistant. For instance, a pilot could be brought to perform a diversion given a certain reason. If the scenario corresponds exactly to one of the training stories, the DM does exactly what the story says, using the Memoization policy (a dialogue policy that remembers the stories from the training data). Otherwise, the DM will extrapolate using the Transformer Embedding

Dialogue policy (TED). Therefore, the more representative stories we have, the better the extrapolation works (see Figure 1).



```
- story: diversion path 1
  steps:
  - intent: intent_divert
  - action: action_reason_exists
  - slot_was_set:
    - reason_exists: false
  - action: action_update_reasons
  - intent: intent_inform_reason
  - action: action_fetch_solutions
  - intent: intent_show_more
  - action: action_fetch_more_solutions
  - intent: intent_choose_option
  - action: action_select_solution
  - intent: intent_insert_flpn
  - action: action_insert_flpn
```

```
- story: diversion path 2
  steps:
  - intent: intent_divert
  - action: action_reason_exists
  - slot_was_set:
    - reason_exists: true
  - action: action_fetch_solutions
  - intent: intent_choose_option
  - action: action_select_solution
  - intent: intent_insert_flpn
  - action: action_insert_flpn
```

Figure 1: Illustration of two examples of a diversion scenario in RASA format upon which model will be trained. RASA lists user utterances under the "*intent*" key, actions executed by the virtual assistant under the "*action*" key and the slot events under the "*slot_was_set*" key. Note that slots are the virtual assistant's memory.

TEST BENCH

The decision assistant CLEA (Cockpit coLlaborative Embedded Assistant) is a task-oriented dialogue system. Its aim is the successful simulation of a conversational agent to help pilots achieve a certain task, such as performing a diversion. The RASA framework used in this study (Bocklisch, 2017) distinguishes the following modules: an NLU module (Natural Language Understanding) which handles user intents, a Dialog Manager which tracks conversation state and predicts the next action to be taken. The output is produced through an Action Server, which ensures in particular Natural Language Generation. Note that the RASA framework does not include statistical natural language generation, system utterances are simple templates (Vlasov, 2018). The modular approach allows us to improve the different modules separately. We will focus on using both ML based predictions and state machines for the DM, and keep the other components fixed, with only minor adjustments (see Figure 2).

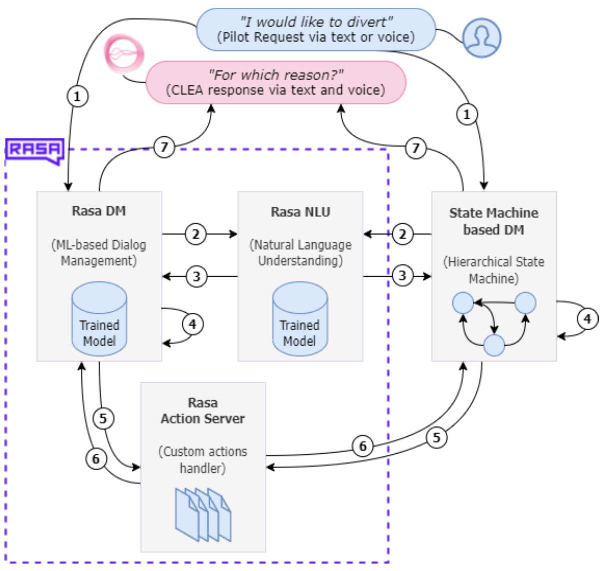


Figure 2: Architecture overview of machine learning based DM vs state machine based DM

The DM controls the flow of the conversation. It includes keeping track of information and choosing appropriate actions based on rules or observations and inferred beliefs. This being said, the modular approach offers us the possibility to create a DM that leverages HSMs in order to compare its performances with the ML-based techniques implemented by RASA.

The figure above outlines the capability to switch between two DMs in a modular dialog system. It means that the system recognizes a dialog act from the pilot as an input. This would be, for example, "*I would like to divert*", uttered either textually or vocally. The DM will make decisions based upon this input and the current system state (step 1). Regardless of the DM's policy regime, NLU receives this input (step 2), in order to map it to an intent (step 3). Therefore, NLU needs to be trained on annotated corpora (Vlasov, 2018).

When using a RASA DM, the prediction of the appropriate action (step 4) will rely on both Memoization and TED policies. When using a State Machine based DM, the choice of the next action (step 4) will be based on manually designed transition diagrams.

As soon as the DM predicts an action, it will trigger the RASA Action Server (step 5) which will execute the right action and return a response to the DM (step 6) that will be uttered textually and vocally to the pilot (step 7). In this case, a suitable response would be "*For which reason?*", in order to know about the reason for the diversion.

REQUIREMENTS AND PRELIMINARY ASSESSMENT

The criteria to compare the candidate approaches are formatted as requirements, specific to the industrial context. According to Airbus method for the design of aircraft systems, system design starts with overlapping operational and functional analysis. Operational analysis produces operational scenarios, which are the starting point of the design of user-system interaction. Scenarios have to be cascaded, enriched and instantiated for several purposes: allocation of functions to assistant skills; evaluation of pilot workload; detailed description of interaction. Our modeling framework shall support a continuity of representations ranging from initial operational scenarios to specialized representations. Thus, the language used to describe interaction should be at least as expressive as the formalisms used for describing scenarios (typically UML activity diagrams, or an equivalent). The capture of operational needs is made iteratively with pilots. The models shall support early simulation, and their representations shall be usable for discussions with end users. Regarding validation, the models shall support early simulation. From the early stages of development, it should be possible for final users to imagine what would be the actual interaction with the system in operations. On verification, it will be hard to cover any possible operational situations through simulation with final users. Other methods should be available to verify the dynamic requirements, for example, co-simulation of system specification and human operations, or formal methods. Implementation of the dialog manager shall be compatible with the resources of the target avionic platform, including not only hardware resources, but also the operating system and the middleware. The development and implementation of Dialogue Management shall be compliant with applicable guidelines from certification authorities (EASA, 2021), in particular regarding learning assurance and explainability. After the entry into service, it will still be needed to reproduce, understand and correct issues and to customize the assistant to the airline's own standards.

WAY FORWARD, DISCUSSIONS

The next steps of the study (before the ICCAS), will finalize the benchmark by modeling the same decision assistance function according to the two candidate approaches, and compare them against the criteria listed in the previous section. The models (either state machines or trained neural networks) will evolve all along a shortened development cycle: initially the model will only support a few normal simple use cases, and then be enriched with marginal and more complex cases. The effort to implement those evolutions will be compared, as well as the performances of the simulations obtained from those models. This evaluation will include exposure of the models to pilots.

Finally, the study will conclude with informed recommendations for the organization of the development of decision assistance functions for the next Airbus aircraft. We keep the possibility to recommend hybrid methods, for example using state machines to generate stories which can be generalized by machine learning, or using either one or the other interaction technique for different use cases.

REFERENCES

Bocklisch, T., Faulkner, J., Pawlowski, N., Nichol, A. (2017). Rasa: Open Source Language Understanding and Dialogue Management.

Bui, T. (2006). Multimodal dialogue management-state of the art. Surface Science - SURFACE SCI.

Degani, A., Heymann, M., Shafto M. (2013). Modeling and formal analysis of human-machine interaction.

EASA. (2021). EASA Concept Paper: First usable guidance for Level 1 machine learning applications. European Union Aviation Safety Agency, proposal issue 01, April 2021.

Goronzy S., Mochales R., Beringer N. (2006). Developing speech dialogs for multimodal HMIs using finite state machines.

Henderson, M., Thomson, B., Young, S. (2013). Deep neural network approach for the Dialog State Tracking Challenge.

Metalinou A. (2020). Science innovations power Alexa Conversations dialogue management. Amazon Science.

Nicol A. (2017). A New Approach to Conversational Software. RASA blog.

RASA Open Source (2021). https://rasa.com/open-source.

Roy, N. (2000). Spoken dialogue Management Using Probabilistic Reasoning.

Vlasov, V., Drissner-Schmid, A., Nichol, A. (2018). Few-Shot Generalization Across Dialogue Tasks.

Vlasov, V., Johannes E., Mosig, M., Nichol, A. (2020). Dialogue Transformers.

Williams, J., Zweig, G. (2015). End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning.

Winograd T. (1986). A Language/Action Perspective on the Design of Cooperative Work. Conference on Computer-Supported Cooperative Work, pp. 203-220.

Yannakakis M. (2000). Hierarchical State Machines, Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, pp.315-330.

Zamanirad S. et al. (2020). State Machine based Human-Bot Conversation Model and Services. CAiSE 2020: 32nd International Conference on Advanced Information Systems Engineering, Jun 2020, Grenoble, France. pp.199-214.