From Process Models to Chatbots

Anselmo López, Josep Sànchez-Ferreres, Josep Carmona(✉), and Lluís Padró

Process and Data Science Group, Computer Science Department,
Universitat Politècnica de Catalunya, Barcelona, Spain
{anselmol,jsanchezf,padro,jcarmona}@cs.upc.edu,
http://www.cs.upc.edu/~pads-upc

Abstract. The effect of digital transformation in organizations needs to go beyond automation, so that human capabilities are also augmented. A possibility in this direction is to make formal representations of processes more accessible for the actors involved. On this line, this paper presents a methodology to transform a formal process description into a conversational agent, which can guide a process actor through the required steps in a user-friendly conversation. The presented system relies on dialog systems and natural language processing and generation techniques, to automatically build a chatbot from a process model. A prototype tool – accessible online – has been developed to transform a process model in BPMN into a chatbot, defined in Artificial Intelligence Marking Language (AIML), which has been evaluated over academic and industrial professionals, showing potential into improving the gap between process understanding and execution.

1 Introduction

Formal process modeling notations are ubiquitous in organizations. They precisely describe a business process, using a graphical notation that has a formal execution semantics, amenable for automating certain tasks of the underlying process [3]. These notations, among which Business Process Model and Notation (BPMN) is a prominent example, are not always suitable or understandable by any actor involved in the process. A good example is a logistic processes, where several agents are required, ranging from agents to transport the goods, down to accountants that keep track of the finances of the whole process.

Hence, one cannot assume always that all the actors of a process would be able to understand a BPMN model, in order to know what they need to do for the successful execution of the process. The fact that digital transformation aims at a better maturity and elicitation of an organization' processes [7], would only contribute to increasing the complexity and size of the process repositories in organizations, which in turn causes a pressure on process' actors. The main goal of the work of this paper is to alleviate this pressure. A similar observation and motivation was presented in the seminal work to convert a BPMN model into a textual description [6], from which this paper shares some parts of the methodology proposed.

© Springer Nature Switzerland AG 2019
P. Giorgini and B. Weber (Eds.): CAiSE 2019, LNCS 11483, pp. 383–398, 2019.
https://doi.org/10.1007/978-3-030-21290-2_24

In this paper we are inspired by a trend seen in the last few years in online services. Often, these sites have a section called *Frequently Asked Questions*, where users can read some solutions to common problems. Sometimes, these pages also have guides to execute some complicated processes or tasks. The main problem is that users have to search for their solution through all the web content, which is often a tedious task. That is why companies are using alternatives to help their customers [15]. One of the most implemented options currently is the *conversational bot* or *chatbot*[1].

Chatbots allow a user to query a complex content, so that a more human interaction with it is enabled. Moreover, the user is relieved from the burden of searching for a solution, which is now a task carried out by the chatbot.

In this paper we present a methodology that takes as input a BPMN model, and generates a chatbot aimed to guide a process actor through the modeled business process. The actor can be guided step-by-step through the process, ask questions about who should perform certain task, or to whom should a document be sent, etc. In this way, a more flexible process interaction is envisioned at a very low cost, since using the methodology proposed some of the processes of a process repository can be transformed into chatbots. The methodology has been validated over 33 individuals, both from academy and industry.

The methodology proposed relies on *script-based dialog management* [16], in which the dialog state determines what is the system expecting at a given moment, and the user utterance will determine the system's answer and the transition to a new dialog state. We generate the finite state dialog automaton from the BPMN structure, and the system utterances from the textual components of the model (task labels, pool and swimlane names, ...), and we add additional states and transitions to deal with user questions about actors (e.g. who should do a task) and objects (e.g. to whom a document must be sent).

The organization of this paper is as follows: next section provides a simple example to illustrate the contributions of this paper. Then, in Sect. 3 we provide the necessary background to understand the methodology that will be presented in Sect. 4. In Sect. 5 we describe the prototype tool implementing the methodologys of this paper, which is then validated in Sect. 6. Finally, Sect. 7 summarizes the paper milestones and reports future challenges ahead.

2 Motivating Example

To illustrate the contribution of this paper, Fig. 2 shows an interaction with the chatbot obtained by applying the methodology proposed on the simple process model shown in Fig. 1[2]. The interaction is shown in the following page.

By a careful look at the interaction, one can see the main ingredients of the methodology described in this paper. First, Natural Language Processing (NLP) is required to analyze the text in the different elements of the BPMN model.

---

[1] One can see an example for the case of computer repair at http://www.toshiba.co.uk/generic/yoko-home/.

[2] This example was found in the Signavio Academic Initiative pool of BPMN models.

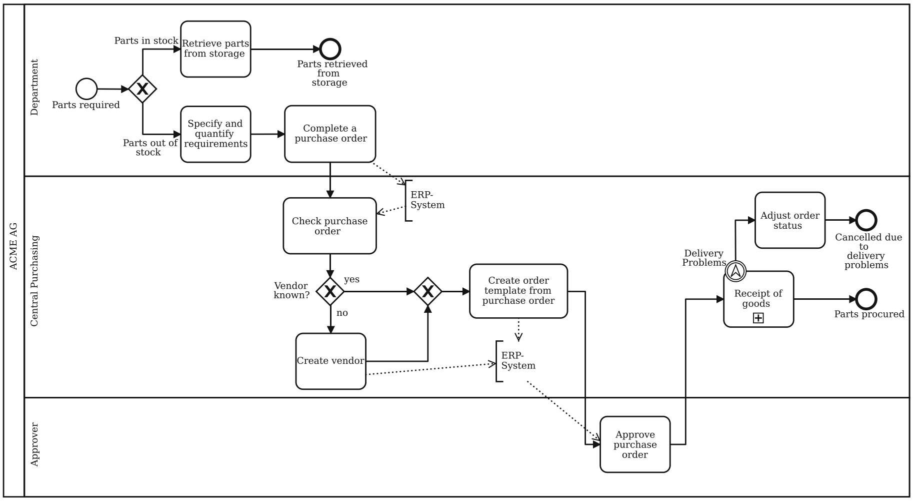


Fig. 1. BPMN representation of the 'procure parts' business process

Second, a tailored finite state automata representation of the underlying process model is used, so that the conversation state is unambiguous depending on the previous questions. Finally, natural language generation is used, so that sentences are created to describe the task to the user in a human-readable manner: see for instance the third-person form when explaining the tasks performed.

3 Preliminaries

3.1 Process Modeling

As it has been already acknowledged in the introduction, formal process notations are an important part of any digitalization initiative, since they open the door to an unambiguous and focused (process) automation. A recent article reports three different process management levels, ranging from multi-process management, to the classical process management level, down to process instance level [8].

Process models can be created using a variety of modeling languages, such as Petri nets, Event-Driven Process Chains (EPCs), and BPMN. Although we focus in BPMN, the contributions of this paper are independent of the specific notation used to define a process model. In particular, we focus on BPMN 2.0, notation created as a standard for business process modeling. BPMN has three different kinds of elements. First, the main elements are the nodes in the diagram, which may belong to three different types: Events, which represent that something happens; Activities, which represents some task that is performed; and Gateways, which split or join flow control. Second, the notation has different edges to connect nodes. A solid line indicates the process workflow, while

Bot: Hi, which process do you want to see?
You: I'm interested in the procure parts process.
Bot: The process begins when the parts are required by the Department. What do you want to do now?
You: Who checks the purchase order?
Bot: The Central Purchasing checks the purchase order.
You: Follow process.
Bot: Parts out of stock/Parts in stock ?
You: Parts out of stock.
Bot: The Department specifies and quantifies the requirements.
You: Next.
Bot: The Department completes the purchase order.
You: Next.
Bot: The Central Purchasing checks the purchase order. Vendor known?
You: Yes.
Bot: The Central Purchasing creates the order template from purchase order.
You: Who did you say before that checks the purchase order?
Bot: The Central Purchasing checks the purchase order.
You: ok.
Bot: The Approver approves the purchase order.
You: Next.
Bot: After that, the Central Purchasing executes process called "receipt of goods".
You: Next.
Bot: Then, the process is finished.

Fig. 2. Example of dialog with the bot generated for the "procure parts" process.

dashed lines represent messages sent between process participants[3]. Finally, there are organization elements such as lanes that contain activities performed by the same participant, and pools, that group several related lanes.

3.2 Dialog Systems

Dialog is the most natural way for humans to communicate, and since the dawn of computers, researchers have aimed to devise ways to communicate with machines as we do with people. From Eliza [19] – the first reactive chatbot – to modern assistants like Siri, Alexa, or Google Assistant, dialog systems construction still strongly rely on large amounts of human intervention, to establish which topics the chatbot should be aware of, and provide useful answers to.

Regardless of whether the dialog interface is oral or written, traditional dialog systems are tailored to a specific task (e.g. helping the user to buy a plane ticket, post a claim for a wrong product, etc.) since the system requires a precise definition of domain concepts and actions to execute depending on the user input. For this reason, they usually are expensive to develop, and not easily customizable to new application domains. This is also the case of modern personal assistants.

---

[3] Dashed lines can also represent data associations.

On the other hand, there are the so-called *recreational* (also known as *conversational*) chatbots which do not target a specific task, but only aim to entertain the user, or to win a Turing's Test competition [18,19].

Dialog systems typically consist of four main components:

- User input processing and understanding: Takes care of processing the user input (which may be speech- or text-based, or even multimodal) and extracting the relevant information and intention.
- Dialog manager: Keeps track of the dialog state, and decides how to update it, and which tasks should be executed at each moment.
- Task Manager: Deals with the back-office operations required for the dialog goal (retrieving information from a database, purchasing tickets, booking reservations, etc).
- Output generator: Produces the appropriate answer or feedback (speech, text, or multimodal) to be sent to user.

Each of this components may be realized at different levels of complexity: Input processing may range from a simple keyword matching on the user text to an advanced Natural Language Processing system. Dialog managers can follow a simple stateless reactive pattern, be based on finite state automata or more complex state-keeping structures, or rely on advanced Machine Learning methods (which require lots of annotated data – actual dialogs – relative to the target domain to be trained). Task Manager – which is missing in non-task oriented dialog systems – is the most domain-dependent component, and must be taylored for each application. And finally, output generation can be approached with techniques ranging from basic pre-written fixed sentences or patterns, up to complete Natural Language Generation systems.

See [1,5] for more details on dialog systems architectures and technologies.

3.3 Natural Language Processing and Generation

Apart from the internal logic or domain-related reasoning that a dialog system must carry out (e.g. access a database to extract available flights matching user's needs, decide which may be most useful, etc.), a crucial part of the dialog is understanding user utterances.

For that, NLP tools are required in order to convert the text spoken or written by the user into structured data that can be used by the system.

In our case, we are generating a chatbot from a BPMN model. For that, we need to extract information from the language components in the model – basically the task labels and pool and lanes descriptions – and for this we also resort to NLP tools to extract the actions being described in the labels, the agents who perform each action, and the objects upon which action is performed. The way we extract this information follows a similar strategy to the one presented in [6].

Another important component in any dialog system is that in charge of generating the system reply that will be sent to the user. Ideally, the system utterance should sound natural, avoid reiteration of already shared information, use a varied set of language structures and lexica, etc. This is addressed by a subfield of NLP known as Natural Language Generation (NLG), that given a semantic representation of the concepts to be expressed, generates the appropriate sentences. NLG is used not only to generate system replies in dialog systems, but also in automatic document generation, either to generate reports from raw from data (*data-to-text* NLG) or from other texts (*text-to-text* NLG) (e.g. automatic summarization).

NLG can be approached at different complexity levels, depending on the task and on the expected results. Simple dialog systems often use pre-canned sentences (which may contain wildcards that are appropriately replaced). A varied set of pre-canned sentences for each situation, from which an answer is randomly chosen when needed, may be enough to avoid a too repetitive user experience.

However, for more advanced NLG applications, complex architectures may be needed. Main steps in a NLG system involve: Determining what to say, planning the structure of the generated text or document, choosing the words to be used, generating the sentences expressing each concept, aggregating or merging several sentences in one to avoid redundancy, introducing pronouns to refer to entities previously mentioned, and finally, realize all that in appropriate and grammatical sentences. More details on NLG techniques can be found in [14].

In our process model scenario, we can not resort to pre-canned text, since each input model may be different. Given that our generated dialog has one state for each model task (see Sect. 4) we apply the *realization* step to obtain a sentence describing the task, and then we use this generated text as a pre-canned pattern at execution time.

4 Chatbot Generation from BPMN

To achieve our goal of generating a dialog agent from a process model in BPMN, we first define which kind of interactions the user is expected to have with the system, namely:

- Ask who is the actor performing any task.
- Ask to who (from who) is a message or a data object sent (received).
- Be guided step-by-step through the process:
    - Find out what is the next task to be executed (or a list of possible tasks, if several are possible) either by a particular actor or in the general process
    - Be asked to provide information when exclusive gateways are reached and be guided into the appropriate branch
    - Be informed when the process ends for a particular actor, or as a whole.

The purpose of these interactions is the use cases that may arise from this work, i.e., helping users to perform tasks of a process model. This type of interactions was required in a short collaboration we had with a process modelling software company. Other types of interactions are left for future work.

Given the expected flows of the dialog, we build a finite state automaton (FSA) that encodes the interactions and conversation flows that we focus in this paper.

The utterances that the system will produce when reaching each state in the FSA are generated analyzing the meaning of the text instances in the BPMN model (task labels and pool/swimlane names), and then feeding this semantic representation into a NLG system.

Also, a variety of patterns to match and interpret user response at each state are generated from model text, plus some general expressions valid for any process (such as "what is the next task?" or "end this conversation").

Once the conversation FSA has been generated, it is encoded into AIML [17], so it can be executed in any available AIML interpretation engine. Figure 3 shows the main steps in the generation process, detailed in the following sections.

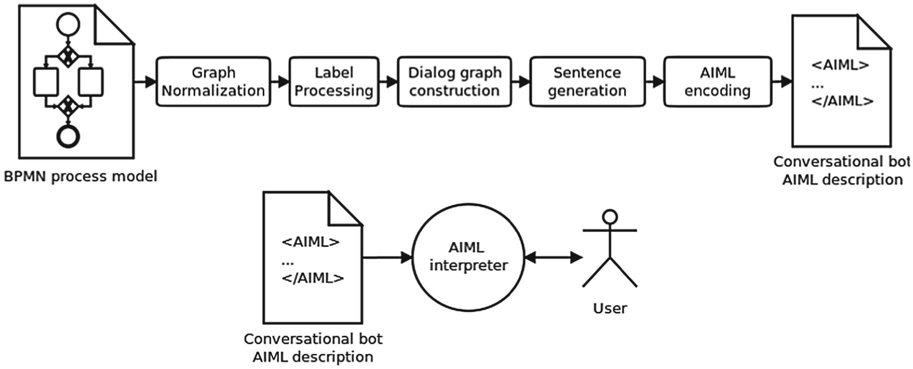


Fig. 3. Chatbot generation process stages (top). Once the chatbot description has been generated, it can be executed by AIML interpreter to interact with the user (bottom).

4.1 Graph Normalization

We start from the BPMN file, and we parse its XML format in order to load the process graph. This graph may require some normalization step, in order to ensure that all blocks in the graph are well-formed. In our case, we aim at having a BPMN that can be partitioned into *Single-Entry Single-Exit* (SESE) components [13]: for instance, the activities `P1`, `P2` and `P3` together with the two adjacent parallel gateways form a SESE in Fig. 5(a). Several transformation techniques can be applied in case a process model is not well-formed (e.g., [12]). Hence, in the rest of this paper, we assume the process is well-formed.

4.2 Label Processing

Once the graph is normalized, we have to collect the linguistic information of model labels. We use FreeLing[4] library [10] to desambiguate the part-of-speech of the label text, and to run it through a custom grammar that extracts the action, the object, as well as other complements. The subject is usually ommitted in the task label, so it is retrieved from the pool or swimlane name.

For instance, the label *Retrieve parts from storage* in swimlane *Department* in Fig. 1 would produce the semantic structure in Fig. 4.

[`action:` *retrieve*,
`subject:` *department*,
`object:` *parts*,
`complement:` *from storage*]

Fig. 4. Semantic structure produced by NLP analysis of the sentence *Retrieve parts from storage* in swimlane *Department* from Fig. 1.

We use a custom grammar and not a general purpose natural language parser such as those provided by FreeLing or other similar library because of the particular structure of model task labels: Task labels are commonly written in simple patterns action-object (*retrieve parts*), or object-nominalized action (*parts retrieval*) with sometimes some additional complement(s) [11]. Also, the subject is usually ommitted, which causes general purpose PoS taggers and parsers to fail more often. Having an ad-hoc grammar allows us to (1) control precisely which patterns should be detected, and (2) feed the parser with k most-likely PoS annotations from the tagger to find out if any of them matches the expected patterns, thus recovering from errors in the tagging step that would lead to wrong parsing results.

4.3 Dialog Graph Construction

Next step is generating the dialog graph, that is, the FSA that encodes all the possible dialog flows. This is a typical architecture followed by many simple chatbots, specially those based on AIML. The dialog graph consists of a set of states and transitions between them. Transition from one state to the next depends on the user utterance.

Definition 1. *A dialog graph FSA is a tuple,* $(Q, \mathcal{T}, \delta, A, \Omega)$*, where:*

Q *is a finite set of state nodes,*
$\mathcal{T}$ *is the set of all possible text utterances emmited by the user,*
$\delta : Q \times \mathcal{T} \rightarrow Q$ *is a transition function that given the current state* $q \in Q$ *and a text utterance* $t \in \mathcal{T}$ *computes the destination state,*

---

[4] http://nlp.cs.upc.edu/freeling.

$A \subseteq Q$ is the set of start state nodes, and
$\Omega \subseteq Q$ is the set of final state nodes.

Note that the transition function δ does not work on a closed alphabet as in normal FSAs. Function δ may range from a simple set of regular expressions performing pattern matching on the user sentence, to a highly sophisticated language analysis system using the latest Artificial Intelligence techniques. In our case, since AIML supports only regular expression based transitions, we restrict ourselves to that approach, though with some extensions provided by the used interpeter (see Sect. 4.5).

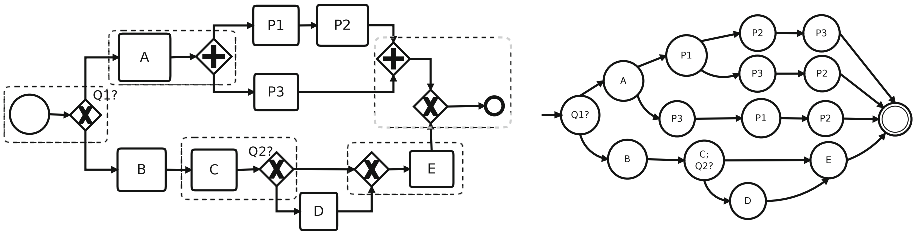


Fig. 5. Initial dialog state graph (right) corresponding to a BPMN model (left). Dotted lines show how split (join) gateways are fused with preceeding (following) elements. Note the expansion of the parallel block into all its possible paths. Self-loops are added later to handle questions or commands valid in any state.

The created dialog graph has a structure that resembles the original BPMN graph, but with some differences to make it suitable for dialog control:

- **Join gateways:** In the BPMN semantics, join gateways describe the point where the branches of a previous split gateway are merged. This kind of node makes no sense in a dialog flow (it would be confusing that the system uttered "Now there is a join. what do you want to do?"). Thus, this kind of nodes are removed from the graph, and its entering edges are associated to the following element.
- **Parallel blocks:** A parallel block consists of the flow elements between a split and a join inclusive gateway. In BPMN, parallel block are interpreted as meaning that the involved tasks may be executed in any order. To account for this behavior in the dialog graph, we create a path in the dialog graph for each valid permutation of the tasks in the parallel branches. In this way, the user can choose the order in which she wants to perform the tasks[5]. Notice that as commented in Sect. 4.1, our strategy to transform parallel blocks (see bellow the formalization of the algorithm for this specific part) assumes that

---

[5] In case of a parallel block consisting on a large amount of parallel tasks, in principle it is not needed to offer all the permutations in the dialog graph, if that contributes to a state-space explosion: the artifact would be that the user only sees a subset of the possibilities offered by the process model, which in some cases can be acceptable.

all parallel blocks in the BPMN are well-structured (a parallel block is *well-structured* when the number of branches going out the split gateway is equal to the number of branches entering the join gateway).

Figure 5 shows a simple example of the transformation of a BPMN model into a graph dialog that guides the user through the process.

The steps performed to recursively expand the parallel blocks and generate the corresponding dialog graph fragment are now overviewed. First of all, a depth-first search traversal is performed to detect split parallel gateways. When one is found, a new parallel block instance is pushed onto a general stack. The stack contains parallel blocks in depth-first order, because we need to guarantee the correct transformation of internal parallel blocks at every depth. Within a particular parallel scope, all the nodes encountered are added to the corresponding parallel branch. If the node is a join gateway, then all the active open branches of the containing parallel block instance are closed, and then the parallel block with the new expanded instance is replaced. For every parallel block detected, we check if there is any parallel block inside. If there is one, we call the function for that node. If there is no block, all the permutations of the branches of the selected block are created in the corresponding newly created FSM fragment. Then, these permutations are connected to the rest of the dialog graph.

Once the control-flow is completely transferred to the dialog graph, the last step of the construction is to also transfer the additional information contained in the BPMN model: messages, actors and data objects. These mainly correspond to self-loops on any conversation state, where information is reported to the user while retaining the conversation state (e.g. the user may ask *who did you say before that checks the purchase order?* even when the conversation is not in the state corresponding to this task).

4.4 Sentence Generation

The dialog graph at this point has the definitive structure, but sentences that will be emmitted by the system at each state have not been generated yet. To generate these sentences, we proceed in consecutive stages.

First, we create the syntactic specifications for each node. This step uses the semantic structures generated during label processing (Sect. 4.2). Using these annotations – and depending on the BPMN element type they correspond to – a syntactic structure is created with the appropriate characteristics (kind of sentence – affirmative, interrogative –, verb features – tense, person, ..., modifiers, etc.) Note that some node types require a special treatment. For instance, at the process start node, the sentence will be headed by the text *The process starts when*, to give the user a better context information. Also, exclusive gateways will be generated as questions and not as affirmative sentences.

Each dialog node can have more than one syntactic structure. Also, the order of the structures can affect the way sentences are generated. The syntactic structures are provided to the *realization engine*, a module that applies syntactic, grammatical and morphological rules to produce a correct phrase with the requested features.

We use the realization engine provided by SimpleNLG[6] library [4], an open-source project that uses basic English lexicon and grammar to transform the input into an appropriate sentence. One of the benefits of SimpleNLG is its potential to be adapted to other languages, using existing linguistic resources and performing some code adaptation.

SimpleNLG provides classes representing different kinds of phrases (verb phrase, noun phrase, prepositional phrase, etc). The calling application can instantiate any phrase specifying the desired features. SimpleNLG engine will build the sentences using grammar rules to properly combine the input phrase instances to form a valid syntactic tree. In our case, we build the phrases using the semantic structures previously created and we use them as required by the node type. Once the phrase instances are created, they are sent to the realization engine to obtain a full sentence.

The realization engine follows several steps to build the final sentence: First, the syntactic rules are applied to obtain the post-syntax tree. This decides, for example, the appropriate order for the words in the target language. Then, the morphological transformations – like selecting the correct determiners or the verb tense – are applied on the obtained tree. Finally, the last step is the orthography function, where sentence punctuation is revised and corrected. If there is some special format required, it is applied after these steps.

Once the sentences for each graph node are generated, we use them as basic information to create the message sentences and the questions:

When the model contains a message element, the user is asked to choose between continuing with the next task in the current lane, or to follow the message and see which task the message recipient will perform[7]. Message information is often encoded in the task originating it, and not in the message element itself, thus the generator needs to check both possibilities and decide which is the right text to use to generate sentences relative to message sending/receiving.

We also generate possible questions about the elements on the process. We resort to the same realization engine to produce questions about who is the responsible for each task, which is the object of an action, or who is the sender/recipient of a message. After some generalization to allow for variations, these questions are included in the set of regular expressions recognized by function δ. Also, this nodes are associated with the related task, so after asking, e.g. *who checks the purchase order* and getting the answer, the user may decide to follow the process from that point, or to remain in the current state.

---

[6] https://github.com/simplenlg.

[7] This can only be done if the information is present in the original BPMN model.

4.5 AIML Encoding

Once the dialog graph is complete and all the needed text has been generated, the dialog can be exported to the desired format to be interpreted by a chatbot engine.

We use *Artificial Intelligence Modeling Language* (AIML) standard because it is the conversational bot definition format most widely used. This XML-based format builds on the concepts of *topics*, which correspond to dialog states, and *categories* to represent the expected transitions from each state. Each category specifies a *pattern* – a regular expression to be matched with the user input, and a *template* providing the answer the bot must emit and the new state to transition to. Since AIML basically describes extended FSAs, it is straightforward to convert our dialog graph into this format. AIML *patterns* allow for the use of wildcards that will match zero or more words in the user input, as well as *sets*, that allow specifying that a word in the user utterance may be any of a given list. We use both this mechanisms to add flexibility to user sentence interpretation, allowing for synonyms, or for extra words inserted in the user input. We pre-encode our *sets* in general synonym dictionaries extracted from WordNet [9].

AIML also supports some features over a pure FSA, such as the possibility of having internal variables to store any relevant information, that may be needed further along in the dialog (e.g. to store some user-provided information such as her name, or some other internal information not encoded in the state). In future versions of our bot generator this could be used, for instance, to ask the user which process role she wants to play, so when describing tasks executed by the selected role, the system would output e.g. *You check the purchase order* instead of *Central Purchasing checks the purchase order*.

4.6 AIML Interpretation

Once the AIML dialog definition file has been generated, it can be executed using any available AIML interpretation engine, so a user can actually interact with the bot.

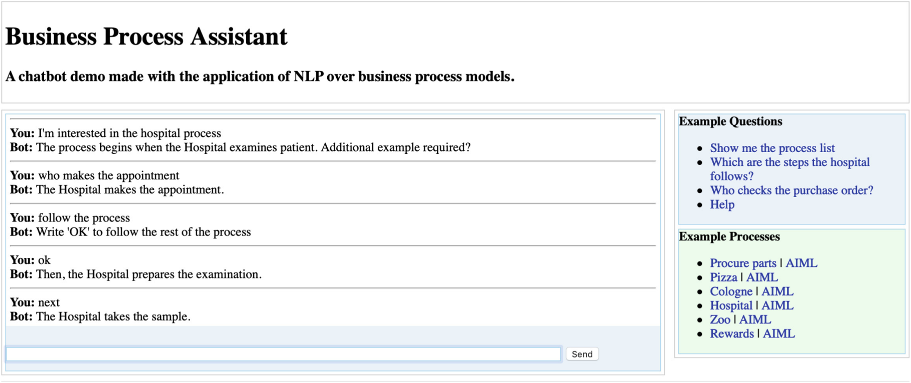
Among the many open source available options, we use *ProgramY*[8]. It is maintained by AIML Foundation (who defines the evolution of the standard), and it is kept in sync with the latest standard updates. Also, it includes some useful additional features, like custom tags o full RegEx support, as well as a variety of front-ends to integrate the dialogs in different environments (standalone, web-based, Telegram, Twitter, Facebook, etc.).

---

[8] https://github.com/keiffster/program-y.

5 Tool Support

The methodology of this paper is available through the NLP4BPM platform [2], accessible at https://bpm.cs.upc.edu/bpmninterface/. Once logged in[9], the user can go to the tab “BPMN to AIML” where a BPMN file can be uploaded and get as a result the AIML corresponding to the created chatbot, applying the methodology of this paper. On the tab “Interpreting AIML” the user can upload the AIML generated to interact with the created chatbot.



Business Process Assistant

A chatbot demo made with the application of NLP over business process models.

You: I'm interested in the hospital process
Bot: The process begins when the Hospital examines patient. Additional example required?

You: who makes the appointment
Bot: The Hospital makes the appointment.

You: follow the process
Bot: Write 'OK' to follow the rest of the process

You: ok
Bot: Then, the Hospital prepares the examination.

You: next
Bot: The Hospital takes the sample.

Example Questions

- Show me the process list
- Which are the steps the hospital follows?
- Who checks the purchase order?
- Help

Example Processes

- Procure parts | AIML
- Pizza | AIML
- Cologne | AIML
- Hospital | AIML
- Zoo | AIML
- Rewards | AIML

Fig. 6. Example of interpretation for several BPMN process models (available at https://bpm.cs.upc.edu/chatbot).

For a fast insight on the contribution of this paper, we have set up an AIML interpreter demonstrating some generated chatbots for a collection of BPMN models, so that a user can interact with them. Figure 6 shows a screenshot of the environment, accessible at https://bpm.cs.upc.edu/chatbot.

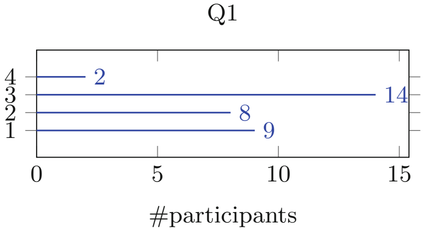
6 Evaluation, Limitations and Use Cases

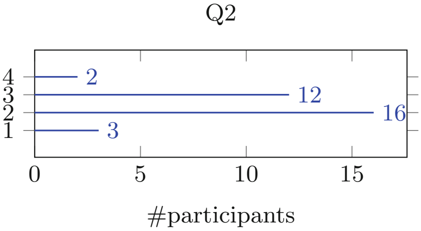
To evaluate the contribution of this paper, we collected feedback of 33 individuals from academia (27) or industry (6). After interacting with the chatbot for a couple of processes, the following questions were answered:

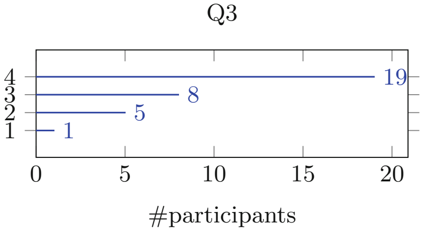
Q1: *How was your interaction with the chatbot ?* (1: not fluent – 4: very fluent)
Q2: *Did the Process Model Chatbot answer your questions about the process?* (1: it did not – 4: it did)
Q3: *Do you see potential for this kind of application in organizations?* (1: no potential – 4: large potential).

---

[9] An anonymous user is temporarily available for review purposes with the username “demo” and password “caisedemo”.







Two more informations were asked, were individuals could provide free text on the following two questions;

Q4: *What did you like/dislike about the tool ?*
Q5: *Do you have any suggestions in order to improve the Process Model Chatbot?.*

From the answers to Q1–Q2, one can see that there is room for improvement in the implementation of our ideas: in both questions, more than half of the answers where on lowest scores. This is an artifact of the limited functionality of the current implementation, which lacks some flexibility and needs to be extended to be able to cover more parts of the process. In spite of this, through the answers to Q3 (81.8% agree on the huge potential of the ideas), we are confident that by improving theses weakness we will be able to come up with a solution that can be of practical use in organizations.

The answers to Q4–Q5 where an interesting source of ideas for improvement and encouragement, but confirmed the limited capabilities of the current implementation. Also, suggestions on use cases where provided, e.g., to help in the training of individuals, to help in the management of process changes, to have a state-aware dialogue between the actors and the process, among others.

7 Conclusions and Future Work

In this paper we have presented a fresh view on the interaction between processes and humans in organizations. By automating the translation between formal model notations like BPMN into conversational agents, a more flexible ecosystem is envisioned. This paper represents the first step towards the ambitious goal of empowering humans in organizations, so that decision-making is facilitated. We foresee multiple directions for future research, among which we highlight:

- Extend the capabilities of the interaction, either by extending the language (in our case, AIML), the types of BPMN constructs to consider, or the interpretation itself. Also, enable the interaction even when the user does not know the main activities of the process.
- Incorporate domain knowledge and/or other perspectives, e.g., data access rights, or security/privacy information.
- Create interactions at the level of process repositories.[10]

Acknowledgments. We would like to thank Gero Decker for drawing our attention to the problem considered in this paper. This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

1. Chen, H., Liu, X., Yin, D., Tang, J.: A survey on dialogue systems: recent advances and new frontiers. SIGKDD Explor. Newsl. **19**(2), 25–35 (2017)
2. Delicado, L., Sànchez-Ferreres, J., Carmona, J., Padró, L.: NLP4BPM - natural language processing tools for business process management. In: Proceedings of the BPM Demo Track Co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September (2017)
3. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56509-4
4. Gatt, A., Reiter, E.: SimpleNLG: a realisation engine for practical applications (2009)
5. Jurafsky, D., Martin, J.H.: Speech and Language Processing, 2nd edn. Prentice-Hall Inc., Upper Saddle River (2009)
6. Leopold, H., Mendling, J., Polyvyanyy, A.: Supporting process model validation through natural language generation. IEEE Trans. Softw. Eng. **40**(8), 818–840 (2014)
7. Matt, C., Hess, T., Benlian, A.: Digital transformation strategies. Bus. Inf. Syst. Eng. **57**(5), 339–343 (2015)
8. Mendling, J., Baesens, B., Bernstein, A., Fellmann, M.: Challenges of smart business process management: an introduction to the special issue. Decis. Support Syst. **100**, 1–5 (2017)
9. Miller, G.A.: WordNet: a lexical database for English. Commun. ACM **38**(11), 39–41 (1995)
10. Padro, L., Stanilovsky, E.: FreeLing 3.0: towards wider multilinguality. In: International Conference on Language Resources and Evaluation, pp. 2473–2479, May 2012
11. Pittke, F., Leopold, H., Mendling, J.: Automatic detection and resolution of lexical ambiguity in process models. In: Software Engineering 2016, Fachtagung des GI-Fachbereichs Softwaretechnik, Wien, Österreich, 23–26 Februar 2016, pp. 75–76 (2016)

---

[10] In https://bpm.cs.upc.edu/chatbot we provide very simple queries for detecting processes in a reporitory. One can think on a more elaborated setting where complex queries can be allowed.

12. Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. Comput. J. **57**(1), 12–35 (2014)
13. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 229–244. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_16
14. Reiter, E., Dale, R.: Building Natural Language Generation Systems. Cambridge University Press, New York (2000)
15. Thorne, C.: Chatbots for troubleshooting: a survey. Lang. Linguist. Compass **11**(10), e12253 (2017)
16. Traum, D.R., et al.: Representations of dialogue state for domain and task independent meta-dialogue. Electron. Trans. Artif. Intell. **3**(D), 125–152 (1999)
17. Wallace, R.: The Elements of AIML Style. ALICE AI Foundation (2003)
18. Wallace, R.S.: The anatomy of A.L.I.C.E. In: Epstein, R., Roberts, G., Beber, G. (eds.) Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer, pp. 181–210. Springer, Netherlands (2009). https://doi.org/10.1007/978-1-4020-6710-5_13
19. Weizenbaum, J.: ELIZA - a computer program for the study of natural language communication between man and machine. Commun. ACM **9**(1), 36–45 (1966)