HHAI2022: Augmenting Human Intellect
S. Schlobach et al. (Eds.)

doi:10.3233/FAIA220195

A Sequence-Based Dialog Management Framework for Co-Regulated Dialog

Florian Kunneman [a] and Koen V. Hindriks [a]
[a] *Vrije Universiteit Amsterdam, Social AI group, The Netherlands*

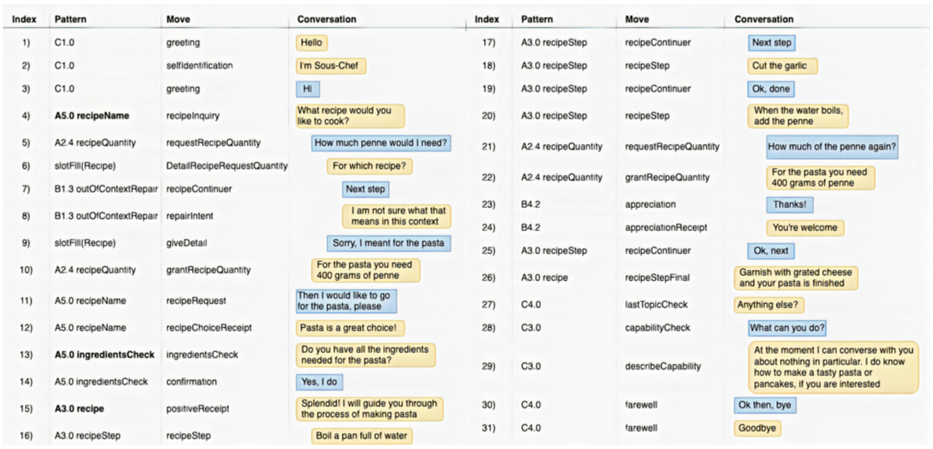
Abstract. We propose SUPPLE (Sequence-Update Pattern-Based Processing with Logical Expansions), a new dialog management framework that takes the core concept of a *dialog sequence* as its main starting point. SUPPLE naturally enables the integration of the flexible and re-usable *conversation patterns* from the Natural Conversation Framework (NCF). Whereas NCF primarily provides a design framework, we developed a dialog engine and authoring framework that builds on the notion of a pattern for specifying sequence structure. In our approach we combine patterns with the key concepts of *update strategies* and *agenda* adopted from the Information State Update (ISU) approach. The main contributions of our work are the introduction of concepts and mechanisms for automatically managing dialog sequences. The framework is implemented as a cognitive agent, and we show through a cooking assistant case study how the agent keeps track of a recipe instruction agenda while allowing for user- as well as agent-initiated sequence expansions. conversational agents to co-regulate the conversation and thus allows for more flexibility.

Keywords. Sequence Updates, Natural Conversation Framework, Mixed-Initiative Dialog, Information State Update, Conversation Patterns, Conversational Agents

1. Introduction

Most dialog management frameworks focus on the dialog moves that are performed and less so on the overall structure of a conversation [1]. According to [2], who propose the Information State Update approach (ISU), the main functions of a dialog manager are updating the current information state given the last move that was performed and selecting the next move to perform (if it is the agent's turn). One recent approach combines dialog state tracking with reinforcement learning, where the dialog state is extracted as the most recent prompt of the system and the utterance of the user, along with the slots that have been collected [3]. End-to-end architectures, finally, directly map the last user utterance to a response formulation [4]. While the conversation history may be incorporated, there is no concrete notion of sequential organisation of the conversation involved.

In contrast, the Natural Conversation Framework (NCF; [5]) focuses more on the structural aspects of conversation and proposes a library of patterns that naturally occur in conversation. NCF also has more to say on how a conversation is managed than most dialog theories. By proposing patterns of conversation moves, it provides a toolbox for designing conversational agents. NCF is primarily intended for guiding the practice of conversational UX design, and does not explicate how conversational patterns could inform a dialog management module in a way similar to how ISU integrates dialog moves.



Index	Pattern	Move	Conversation
1)	C1.0	greeting	Hello
2)	C1.0	selfIdentification	I'm Sous-Chef
3)	C1.0	greeting	Hi
4)	**A5.0 recipeName**	recipeInquiry	What recipe would you like to cook?
5)	A2.4 recipeQuantity	requestRecipeQuantity	How much penne would I need?
6)	slotFill(Recipe)	DetailRecipeRequestQuantity	For which recipe?
7)	B1.3 outOfContextRepair	recipeContinuer	Next step
8)	B1.3 outOfContextRepair	repairIntent	I am not sure what that means in this context
9)	slotFill(Recipe)	giveDetail	Sorry, I meant for the pasta
10)	A2.4 recipeQuantity	grantRecipeQuantity	For the pasta you need 400 grams of penne
11)	A5.0 recipeName	recipeRequest	Then I would like to go for the pasta, please
12)	A5.0 recipeName	recipeChoiceReceipt	Pasta is a great choice!
13)	**A5.0 ingredientsCheck**	ingredientsCheck	Do you have all the ingredients needed for the pasta?
14)	A5.0 ingredientsCheck	confirmation	Yes, I do
15)	**A3.0 recipe**	positiveReceipt	Splendid! I will guide you through the process of making pasta
16)	A3.0 recipeStep	recipeStep	Boil a pan full of water
17)	A3.0 recipeStep	recipeContinuer	Next step
18)	A3.0 recipeStep	recipeStep	Cut the garlic
19)	A3.0 recipeStep	recipeContinuer	Ok, done
20)	A3.0 recipeStep	recipeStep	When the water boils, add the penne
21)	A2.4 recipeQuantity	requestRecipeQuantity	How much of the penne again?
22)	A2.4 recipeQuantity	grantRecipeQuantity	For the pasta you need 400 grams of penne
23)	B4.2	appreciation	Thanks!
24)	B4.2	appreciationReceipt	You're welcome
25)	A3.0 recipeStep	recipeContinuer	Ok, next
26)	A3.0 recipe	recipeStepFinal	Garnish with grated cheese and your pasta is finished
27)	C4.0	lastTopicCheck	Anything else?
28)	C3.0	capabilityCheck	What can you do?
29)	C3.0	describeCapability	At the moment I can converse with you about nothing in particular. I do know how to make a tasty pasta or pancakes, if you are interested
30)	C4.0	farewell	Ok then, bye
31)	C4.0	farewell	Goodbye

Figure 1. Example dialog between user and SUPPLE agent. Yellow boxes are agent and blue boxes are user utterances. Indented utterances mark sequence expansions. Patterns in bold are initiated based on the agenda.

We propose SUPPLE (Sequence-Update Pattern-Based Processing with Logical Expansions), a new dialog management approach that combines the strengths of ISU and NCF - managing conversations in a natural way using patterns and sequence expansion as the core mechanism for update strategies, and including agenda-keeping capacities for initialising and adapting the direction of a conversation. Our approach offers a rich set of design patterns derived from NCF. There is no need for completely scripting or designing the dialog flow, but only a need to create patterns that the conversational agent can select from. This allows for a more flexible conversation, where both user and agent can co-regulate the interaction [6], in order to cover the full range of natural dialogs users may want to engage in [7] and to impose as few constraints on the dialog as possible [8].

To illustrate two common challenges in dialog systems and how they are addressed in SUPPLE, we present an example conversation with a cooking assistant in Figure 1. The assistant walks the user through a (dummy) recipe by following the course of a set of conversation patterns. One of the main challenges in dialog systems is how to support *flexibility*, which can be defined as the extent to which both parties can take the initiative in a dialog. The conversation patterns in the example are either initiated by the agent, often based on convention (opening the conversation, e.g. Fig. 1.1-3) or its agenda (checking whether ingredients are available in 1.13-14, instructing the recipe in 1.15-26), or by the user when asking a question (lines 1.5, 1.21 and 1.28) or thanking the agent (1.23). One of our design goals is to facilitate such a *mixed-initiative interaction* . Dialog systems that support mixed-initiative interaction need to be able to handle complexities due to the numerous and varied directions in which the user might steer the dialog [9]. This means that such systems need to be able to cope with utterances that fall outside of the system's expected dialog sequence.

A second challenge is related to the engineering of dialog systems. Our work aims to address the question of how to *simplify* the engineering of and facilitate *re-use* in designing a dialog system, which is considered a complex task and difficult problem requiring much effort in the literature [10,11]. Our work in this regard is similar to the goals of [12,13], who aim to facilitate authoring dialog and specifying dialog system behaviour. We believe our approach provides for a clear recipe of what a conversational designer needs to do whereas our framework already automates the domain-independent mechanisms provided to a user for navigating through a dialog. The conversational patterns aligned with the conversation in Figure 1 can easily be implemented in our framework to be readily applied by the agent. Although an expert developer can still adapt these rules or fine-tune how they apply, the essence of creating a dialog system for an application domain is selecting the patterns needed and a developer can focus attention on content.

Our work contributes the following:

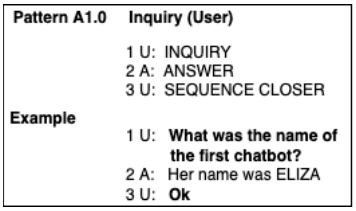
- A new *dialog engine* that enhances co-regulation in conversations by means of *sequence update mechanisms*. These mechanisms enable expanding a sequence with sub-sequences (sub-dialogs), jumping to different patterns, and initiating repair in case of recognition or other problems;
- A *pattern library* that integrates generic conversation patterns for opening, closing, repairing, and others from NCF. This library facilitates *re-usability* and *configurability* across conversational agents of domain-independent dialog capabilities;
- An *authoring approach* that facilitates design of re-usable and easy-to-adapt conversation patterns. This approach aims to reduce the effort for and complexity of developing a mixed-initiative dialog management system [10,11].

2. Related Work

There have been many different frameworks proposed for managing a dialog by a conversational agent. For example, [8] proposes the use of frames, and [14,15,16,17] a plan-based approach to manage a dialog. A shared goal among several of these frameworks is to approximate human-like performance in conversations, with as a prominent aim to enable flexible mixed-initiative conversations that can be shaped by both the agent and user. To this end, [18] incorporate insights from the field of conversation analysis by structuring dialog with conversational fragments and adjacency pairs for an interactive toy. Like them, we propose a mixed-initiative framework that is inspired by the same concepts, but we add sequence expansion as a core mechanism to our dialog engine, which considerably enhances the possibilities of interlocutors to take initiative while also maintaining structure in the conversation. [19] models mixed-initiative in their dialog manager as a reward system, where the agent takes initiative when this is expected to be rewarding in the conversation, and likewise allows humans to do so for the same reason. This, however, also reduces the scope of natural conversational directions, as it is difficult to frame many natural sequence expansions in terms of rewards.

Our dialog management framework builds on Prolog for modelling the agent's knowledge about a domain and conversation in general. Likewise, the TRAINS project [14] proposes a logic-based approach to dialog management, using episodic logic with inference and including an abstract execution plan with expectations regarding the dialog acts to perform, which is monitored and re-planned if needed. Our framework also features planning and adaptation functionality, but offers more flexibility by using sequences of dialog acts as central building blocks that can be expanded on by the agent or user.

[13] proposes a dialog management framework based on concepts from programming language theory emphasising the value of the concept of partial evaluation for flexible dialog management. Their main goal is to offer a solution for the combinatorial explosion that needs to be dealt with when the order of utterances in a dialog can become



Pattern A1.0 **Inquiry (User)**

1 U: INQUIRY
2 A: ANSWER
3 U: SEQUENCE CLOSER

Example

1 U: **What was the name of the first chatbot?**
2 A: Her name was ELIZA
3 U: **Ok**

Figure 2. Example of a conversational pattern mark-up (A=agent, U=user) adapted from [5].

very flexible as is often the case in mixed-initiative systems. Our goals are similar but the tools we propose different. One of the main lessons we derive from the work of [13] is that the size of the patterns used in the dialog management framework that we propose should be kept small. The smaller the patterns are that are used the fewer variants there exist. Moreover, having many different small patterns allows for great flexibility in our approach as in principle the key mechanism of sequence expansion allows for any combination of the available patterns.

The RavenClaw framework introduced in [17] aims to support domain-independent conversational skills, and the development of mixed-initiative systems operating in task-oriented domains. It requires the conversational designer to specify a *hierarchical plan* for the interaction. Instead of asking the designer to specify a tree structure at design time, our framework only requires the designer to specify *linear* structures (patterns) which the dialog engine is able to turn into a dialog tree at run-time.

3. Background

We provide some background on the key concepts of the Information State Update approach and the Natural Conversation Framework that our work builds on.

Information State Update The Information State Update (ISU) approach was originally proposed by [20] as a way to specify how dialog moves can trigger a subsequent agent move in a structured conversation. At its core are the concepts of *information state*, *dialog moves*, and *update rules* [2]. The information state, which is represented as a record, keeps track of all aspects relevant to the conversation, such as the conversation history, the knowledge and beliefs of the agent, and the agenda which guides the agent's targets in the conversation. Information states are updated by update rules which are triggered by the current state and dialog moves performed by the dialog participants [21].

Natural Conversation Framework The Natural Conversation Framework (NCF) is proposed by [5] and based on insights from Conversation Analysis [22].

NCF is aimed at improving the user experience of conversational interfaces by proposing a pattern language similar to what is used in interaction design [23]. The pattern language consists of conversational patterns as common sequences of social actions which are similar to the dialog moves in ISU. The patterns can be used to shape the dialog behaviour and capabilities of a conversational agent. NCF is proposed as a framework to be used by development teams of conversational agents. The pattern language may be applied in combination with any platform, but is particularly aimed at those that support the Intent-Entity-Context-Response paradigm, which we assume as the default approach for natural language understanding here (see also Section 4.3).

The patterns in [5] are specified as a sequence of moves by either user or agent and are accompanied by an example conversation. Each pattern is given a code so as to facilitate re-usability. Figure 2 provides an illustration of a typical pattern, where a question first is posed that is followed by an answer and a sequence closer. [5] describes many of such patterns, categorised into Conversational Activity, Sequence-level management and Conversation Management UX patterns. These conversational patterns can inform an agent about the direction that a conversation may follow, while offering the flexibility to follow different paths and allowing for turns that switch from one pattern to another.

Patterns established by conversation analysis aim at identifying the sequential expectations that are raised and oriented to by participants - inquiries are typically followed by an answer [24]. They are deliberately kept short to enable sequence expansion, for example, when the question is not completely understood and a paraphrase is given instead of an answer, which may in turn be confirmed after which the answer is given. By means of sequence expansion humans can maintain common ground and satisfy conditions needed for closing an original sequence. The ambition of our framework is to automate these mechanisms and provide a method for specifying patterns by a dialog author and used as sub-sequences that relate to the main active conversational sequence.

4. SUPPLE

In SUPPLE, we adopt and integrate concepts from both ISU and NCF. From ISU we take the concepts of *information state* and *agenda*, following [2], which can be used to provide a solid way for keeping track of the conversation, and for guiding subsequent agent moves to give direction to a conversation. From NCF, we borrow concepts for modelling dialog structure. In particular, we include as first-class citizens in our approach the concepts of a *sequence (expansion)* and *pattern* from NCF.

ISU allows an information state to vary per dialog theory and specifies various components that can be used to model such a state. In SUPPLE, the agent maintains an information state that consists of an *agenda*, the *session history*, and a *memory* (beliefs in the terminology of ISU). The session history represents the (in)complete sequences of a conversation and the progress made in the conversation thus far. We store the complete session and not only the last dialog move. The memory extracts and keeps useful information provided by the user about entities up-to-date (e.g., which recipe a user selected). In addition, patterns are stored in the agent's knowledge base as static information about basic dialog conventions.

The architecture of SUPPLE is depicted in Figure 3 and consists of a *dialog authoring* approach (Section 4.1), a *dialog engine* (Section 4.2), and *user interaction components* (Section 4.3). [1] SUPPLE's *dialog authoring* approach complements NCF's generic design framework by providing concrete guidelines on how to instantiate conversational patterns with domain-specific content. The SUPPLE *dialog engine* defines and implements the mechanisms for sequence expansion and provides support for coordinating the flow of the dialog and communication with other sub-systems and components. User interaction in SUPPLE is supported by means of a user interface (which can be either text or voice-based) for collecting user input and a Natural Language Understanding (NLU)

[1]See `https://bitbucket.org/socialroboticshub/clients/src/master/dialogmngr/` for the source code.

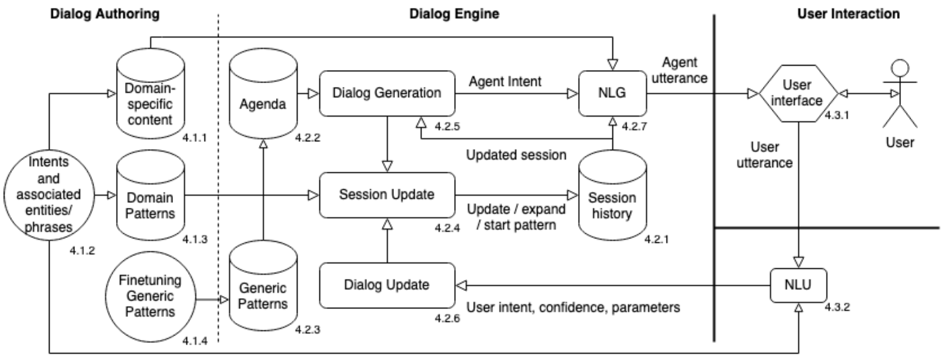


Figure 3. The SUPPLE Architecture (codes next to boxes map to the subsections in this paper)

component based on the IECR paradigm. We assume the NLU component outputs intents (dialog moves), entity key-value parameters, and recognition confidence.[2] The dialog engine and the dialog authoring notation are implemented as a cognitive agent in the GOAL programming language [25] and Prolog [26].

4.1. Authoring Language

We introduce the dialog-authoring notation of our approach to demonstrate its versatility but also the ease with which (at least) a (simple) conversational agent can be specified. A conversational agent can already be developed with only basic knowledge of Prolog. Although there is no requirement to develop advanced capabilities, the use of Prolog (or other KR languages) has the benefit of providing the agent with access to more advanced reasoning capabilities. We illustrate below how the memory maintained automatically by the dialog engine can be queried to select appropriate phrases by the agent. We note that even though the approach we discuss here requires specifying knowledge, this does not preclude the use of automated approaches to extract knowledge about a domain or conversational patterns.

The approach for developing a conversational agent in our framework consists of four key steps: the conversational designer needs to (1) specify domain-specific content, (2) specify intents and associated phrases, (3) specify domain-specific conversational patterns, and, last but not least, (4) fine-tune the agent's behaviour by specifying various parameters for generic patterns.

4.1.1. Specifying Domain-Specific Content A designer is free to specify domain content any way they want by introducing Prolog facts and rules. For example, the fact `ingredient(pasta, penne,'400 grams')` specifies the amount of penne needed for making pasta and `step(pasta, 2, 'Cut the garlic.')` specifies the second step of the pasta recipe. Including the step number here is useful to be able to iterate over recipe steps in the recipe pattern.

4.1.2. Specifying Intents and Associated Phrases A designer needs to specify both user and agent intents, which are the basic building blocks for specifying patterns. Specification of user intents and associated entities depends on the NLU component used. It is the designer's responsibility to optimise the intent recognition from user input by providing

[2]In our current implementation we used Google Dialogflow https://cloud.google.com/dialogflow.

examples of such input. For cases where the NLU component fails, the dialog engine will automatically deploy a generic repair pattern for handling intent recognition failures.

The agent intents name the dialog moves available to the conversational agent. The main task here is to specify intent-phrase pairs of the form `text(recipeInquiry, "What recipe would you like to cook?")` that the agent can use to generate utterances. Generating phrases often requires specifying a rule to be able to access context and domain-specific information. To this end, several predefined predicates are supported by the dialog engine, such as `stepCounter/1` for keeping track of steps that need to be iterated and `keyValue/2` for accessing memory. For example, `text(recipeStep, Txt) :- currentRecipe(Recipe), stepCounter(Cnt), step(Recipe, Cnt, Txt)` uses the domain knowledge about recipe steps illustrated above to retrieve the next instruction text for the intent `recipeStep`. A designer can also define queries to extract relevant information from the session (history). For example, `currentRecipe(Recipe) :- keyValue(recipe, Recipe)` used above accesses memory to retrieve the current recipe.

Finally, a designer needs to provide domain-specific phrases for some of the intents that are part of the generic pattern library. For example, there is a generic pattern for performing capability checks by a user and a designer needs to specify the phrase in Figure 1.29 for the agent intent `describeCapability`.

4.1.3. Specifying Patterns The most important task of a designer is to specify *domain-specific* patterns for structuring the conversation. Patterns specify the order in which intents are expected by the dialog engine and whether the agent or user is expected to generate these intents. To be precise, a *SUPPLE pattern* is a list with an identifier (its name) at the head of the list followed by a list of actor-intent pairs which indicate who is expected to take the turn (either user or agent) and which intent is expected; an actor-intent pair can also be replaced with an agenda management action (Section 4.2). One approach is to specify domain-specific patterns by instantiating some of the patterns discussed in [5]. For example, `pattern([a20recipeOptions, [user, requestRecipeOptions], [agent, recipeOptions]])` specifies a pattern with id `a20recipeOptions` which allows a user to ask which recipes are available and the agent to respond with a list of options. Multiple *variants* of a pattern with the same identifier can be specified which allows for more flexible user interaction. Besides actor-intent pairs, agenda management actions can be used as part of a pattern for, e.g., repeating a sub-pattern. This allows for more sophisticated interactions with a user. For example, `pattern([a30recipe, [agent, recipeConfirm], [agent, repeat(a30recipeStep)], [agent, finalStep]])` instantiates an extended telling pattern where `a30recipeStep` is a sub-pattern that is repeatedly deployed until all steps of a recipe have been made.

4.1.4. Fine Tuning Generic Patterns The dialog engine has a library of generic patterns (including generic intents) which can be re-used that do not need to be specified by a designer. Many of these generic patterns have variants which can be activated or fine-tuned by means of parameters available to a conversational designer. For example, parameters for specifying the agent's name (e.g., `agentName('Sous-Chef')`) and whether the agent takes the initiative or performs a welfare check can be used to make the agent start by self-identifying or not (Fig. 1.2). Other parameters can be used, for example, to make the agent perform a check at the end of a session on whether a user needs anything else (Fig. 1.27), or for fine-tuning the response of repair mechanisms.

4.2. Dialog Engine

4.2.1. Session history This is the core of the dialog engine, in which the running conversation is stored and updated. A *session (history)* is a list of sequences, where each sequence in a session is associated with a pattern. A *SUPPLE sequence* essentially is an (incomplete) pattern where the actor-intent pairs are replaced by actor-intent-parameter triples, agenda management actions, or subsequences (i.e. subdialogs). The parameters in the triples store additional information about entities extracted from the user input during a conversation (e.g., a recipe name). Contextual elements such as the most recent values for entity parameters (overwriting older ones) are also stored in the agent's *memory* for convenient access but only the session maintains the full history of user input and can be queried to retrieve this information. We say that a sequence is *complete* if the sequence without subsequences matches with a pattern, which indicates the sequence has completely executed the pattern. The currently active (incomplete) sequence at the head of a session is used by the dialog generation module to select a next agent dialog move (intent) that matches with the sequence's corresponding pattern if it is the agent's turn.

A session history has a tree-like structure to represent the relation between sequences and subsequences: when an active sequence is expanded with a subsequence, this subsequence becomes part of the active sequence. The information state in our approach thus is not a flat representation of plans or questions under discussion [2] but at runtime is similar to plan-based approaches such as [17,21].

4.2.2. Agenda The agenda is also part of the information state that is updated during the course of a conversation. A *SUPPLE agenda* is an ordered list of pattern identifiers, which specifies the overall "plan" of the agent for carrying out a task. The agent uses its agenda as a top-level schedule in its dialog generation module to structure the conversation, and thus has a goal-setting function. Several *agenda management actions* may be performed on the agenda.

4.2.3. Generic patterns This knowledge base stores patterns and intents that are applicable to most conversational contexts. Examples include patterns for opening and closing a conversation, for sequence closing and repair, and for automatic slot filling.

4.2.4. Session Update This module performs updates to the session history based on the last dialog move that was performed by either user or agent. By design we know that the dialog engine will always select agent moves that match expectations (set by the currently active sequence) and therefore agent moves can simply be added to the active sequence. However, as we cannot control user input to match such expectations and instead want to allow for user initiative, the engine is provided with mechanisms for handling each of the following cases. These mechanisms for sequence management and expansion are at the heart of and are a core contribution of the SUPPLE framework:

- User input matches expected input and contributes to the active sequence (e.g. Fig. 1.3): recognised intent matches the next intent in a pattern associated with the currently active sequence and user input is *appended* to the active sequence to make progress to complete the associated pattern.
- User input does not match expectations but corresponds with a first move in a pattern that can be used to validly *expand* the current sequence (e.g. Fig. 1.5): the active sequence is expanded with a new *subsequence* consisting only of the associated pattern identifier.

- User input corresponds with the start of a known pattern and there is no currently active sequence: a *new sequence* is added to the head of the session history (this only happens if the agent is not set to take the initiative in those cases).
- User input that does not match expectations and is only known to occur later than the first move in any pattern (e.g. Fig. 1.7): this triggers expansion of the active sequence with a subsequence that focuses on *repair* toward the active sequence.

The session update module also handles *turn-taking* and determines whose turn it is from the pattern that matches the currently active sequence. Turn taking may take place at each turn conversational unit, but does not need to. Multiple moves may be subsequently performed by the agent.[3] In case the agent performs consecutive moves, this is in line with either the conversational pattern or agenda (after finishing a sequence, the agent might want to start a new sequence in line with its agenda).

4.2.5. Dialog Generation The Dialog Generation module decides which dialog move the agent should make next based on the ongoing session and its agenda. If a pattern is active with an expected agent intent that can be performed, this intent will be selected. The currently active sequence in a session history thus provides a mechanism for keeping track of discourse obligations [27]. From the last user move the agent can infer if it needs to respond to that move by checking if the pattern that is executed requires the agent to make a move. It may also be the case that the last user move ends a subsequence, in which case the agent can proceed with older obligations, such as the need to still provide an answer after a clarification expansion of the sequence. Note that obligations are not explicitly represented in the system but rather inferred from the session history using the reasoning rules that are part of the agent's knowledge. We assume an agent will always comply with the expectation of the currently active pattern, i.e. perform an utterance with the intent that follows the last intent, or will engage in a clarification or request dialog such as requesting for additional values for entities (slot filling, e.g., Fig. 1.6). Such a clarification or request dialog can be instigated by the agent if a pattern is active while an expected agent intent cannot be performed. If, for example, this is due to lack of information, the agent will expand the sequence with a subsequence to enable the expected agent intent. The Dialog Generation module is also responsible for initiating generic patterns for opening and closing a conversation (e.g., Fig. 1.27) and for executing agenda management actions (e.g., Fig. 1.16).

4.2.6. Dialog Update This module handles incoming user intents from the Natural Language Understanding module (see Section 4.3) and updates the agent's memory with newly received entity values. It matches the intents to the moves that it knows about and handles unknown intents by expanding the current sequence with a repair sequence.

4.2.7. NLG The Natural Language Generation (NLG) module transforms agent intents into a textual or spoken utterance. As natural language generation is beyond the scope of this paper, we provide a simple generic template-based approach for generating natural language phrases that the agent uses to respond to user input (see Section 4.1). In its most simple form, all that is needed is to associate a textual phrase with an intent. In its more advanced forms, language generation can involve filling in phrase parts by means of inferred information from knowledge sources. Alternatively, general purpose natural language generation may be deployed for transforming semantic representations to text.

[3]Multiple moves by the user might yield a conflict with the known conversation patterns. In practice, however, intent recognition is commonly not catered for detecting multiple intents from a single user turn.

4.3. User Interaction and Implementation

4.3.1. User interaction The SUPPLE framework does not make any specific assumptions about user interaction nor the modalities used (e.g., text, speech) other than that such interaction is facilitated through a conversational interface. Currently, the framework supports a textual as well as basic voice-based (with strict turn-taking) interface.

4.3.2. Natural Language Understanding SUPPLE assumes that any utterance received from the user via an interface is interpreted by a separate Natural Language Understanding (NLU) component that outputs a recognised intent, associated entities, and recognition confidence (if any). The output of the NLU component is forwarded to the dialog engine. A range of mature applications are available for Intent-Entity-Context-Response-based dialog modelling that only require a conversational designer to provide example phrases to be able to perform intent recognition by means of exact matching and Machine Learning in the back-end. We make use of Google Dialogflow for the interpretation of user utterances but other similar applications such as RASA [28] may also be used.

4.3.3. Cognitive agent The dialog engine is implemented using the cognitive agent programming language GOAL [25]. Agents have been used before for implementing dialog management, e.g. [29]. For our purposes, it is particularly useful that cognitive agents are rule-based and therefore also offer a natural implementation framework for the update rules and strategies of ISU.

5. Case Study: Cooking Assistant

To showcase the co-regulation and agenda-keeping capabilities of SUPPLE, we designed a cooking assistant example that has the purpose to guide a user through a recipe and help them with questions about the recipe, cooking skills and ingredients. Recipe instruction as a genre has a solid conversation structure in the form of the recipe steps, and likewise requires sequence expansions to enable, for example, clarification of recipe steps.

An example conversation, already briefly introduced and referenced above, is displayed in Figure 1. As before, we refer to lines in the conversation by reference to the figure and index, e.g. 1. 1 refers to the first opening line. Here, we focus on how SUPPLE enables particular mixed-initiative dialog sequences. The patterns and moves are specified in separate columns in Figure 1. The pattern codes are based on patterns discussed in [5]. Codes followed by a name are adjusted to the cooking assistant domain, while codes without a name refer to domain-independent patterns that are part of the generic pattern library adapted from [5]. The recipe that is discussed in the conversation is deliberately kept short to show most conversational mechanisms enabled by SUPPLE, but will not result in a proper pasta meal.

Opening and closing The opening and closing lines 1.1-3 and 1.27-31 are based on generic patterns as they are standard to most conversational genres. These patterns can be agent- or user-initiated and fine-tuned by various parameters (see Section 4.1). The cooking assistant agent initiates the conversation and introduces itself.

The agent has been adapted to perform a last topic check when the agenda has successfully been completed. The dialog engine will therefore automatically trigger the associated C4 pattern and the agent will proceed to a last topic check. In our example, the user expands the sequence with a capability check (also standard to most assistants) after which the user decides to say goodbye, which is then followed by a closing goodbye from the agent. There are multiple variants of the C4 pattern available to handle different user responses. For example, the user could also have given a disconfirmation regarding the last topic check, after which the agent would reply with a wellwish.

Agenda keeping The patterns marked in bold in Figure 1 are part of the agenda that the agent is initialised with. The agent initiates these patterns in order, only when a preceding top-level pattern is successfully completed. That means that the agent will only continue with agenda items (patterns) if all sub-dialogs have either been completed or aborted and the conversation has returned to the main top-level. While the agenda is maintained by the agent, the user has the option to add agenda items (for example, a second recipe) or put agenda items into focus. These actions are not manifested in the example.

Sequence expansion Lines 1.4-11 illustrate the sequence expansion mechanism, which is at the core of the SUPPLE framework. They introduce four different levels of expansions indicated by indentation. The agent and user consecutively take the initiative in the conversation, i.e., initiate a (sub)sequence. These sequence expansions are typical of the mixed-initiative conversations enabled by SUPPLE. The consecutive sequence expansions are in turn closed in reverse order, in line with the tree structure of the session history. This structure enables the agent to keep track of the sequence(s) before expansion and correctly interpret the user answer in 1.11 to a question it asked seven turns earlier.

After the agent asks the user for the preferred recipe in 1.4, the user takes the initiative and replies by asking for the quantity of a particular ingredient in 1.5. By receiving a user intent that is not expected (the agent expects a recipe name as answer to its question) but instead matches with the first intent of another domain-specific pattern, the dialog engine initiates a sequence expansion with this matching pattern. This in turn spurs the agent to take the initiative because it identifies it misses information (slots) by further expanding the sequence with a generic slot filling pattern to ask for additional details so as to be able to answer this reply in 1.6. The user then says something unintelligible in this context in 1.7 even though the NLU component identifies the move as a recipe continuer intent. The dialog engine processes this move as out-of-context, which triggers an expansion with a response of the agent in a new repair sequence 1.8. The user then continues and gives an answer to the agent's question in 1.9. With the additional information collected, the agent can now answer the original question of the user in 1.10, after which the user replies to the agent's initial question by making a choice for a recipe in 1.11. At that point all (sub)sequences have been closed again.

The Repeat Agenda Management Action Lines 1.15-26 illustrate the use of the agenda management capability to repeat a (sub)dialog as many times as needed. This allows for a form of expansion by which the length of a pattern is dynamically decided based upon, in this case, the length of the recipe to be instructed. The A3 recipe pattern initiated in 1.15 is consecutively expanded by the A3 recipe step pattern in lines 1.16, 1.18, and 1.20, matching the first three steps of the recipe. The repeat action iterates over these steps until there are no steps left to instruct. This facilitates conversations guided by an extended and repetitive information exchange, like storytelling, instruction or interviewing. Note that the last step is handled differently to avoid the dialog engine getting stuck on the expectation that a user replies with a continuer.

Contextualised response Lines 1.21-24 illustrate how agent responses are contextualised. In 1.21 the user essentially repeats its earlier question in 1.5 how much of an ingredient will be needed. Whereas at 1.5 it is not yet clear which recipe the user will select, this is different at 1.21 where the selected recipe is already being instructed. The agent at that point is able to retrieve the recipe from memory and therefore does not initiate the slot filling pattern as in 1.6 but immediately is able to answer the question. The subsequent appreciation in line 1.23 does not match the expected continuer which initiates a matching (sub)dialog at the same sequence level.

6. Discussion and Conclusion

We propose the SUPPLE (Sequence-Update Pattern-based Processing with Logical Expansions) dialog framework. SUPPLE addresses the core challenge of enabling mixed-initiative, yet structured dialog with a conversational agent, where both agent and user can co-regulate the course of the conversation and the agent keeps track of the conversation to progress toward certain goals. The SUPPLE dialog engine provides a range of mechanisms for managing conversations by building on the key notion of *sequence expansion*. Re-usability and development effort and expertise are additional challenges which we address by introducing an authoring approach which separates the design of domain-specific conversational patterns from the domain-independent generic patterns and conversational skills that are available to a SUPPLE agent.

A key contribution of our work has been to add conversational management, repair, and other capabilities to a dialog system inspired by the work of [20,2,5]. We thus build on sophisticated theories for dialog updating of information states [20,2], complemented by the work of [5] which focuses on conversational structure. Where [5] mostly addresses conversational design, we have designed SUPPLE as a new approach that provides *automated support for conversational sequence management*. Conversational context is provided by SUPPLE by means of instantiated patterns, i.e. sequences of dialog moves associated with utterances from either user or conversational agent. The mechanism of sequence updating based on processing predefined patterns, which is at the core of the SUPPLE framework, provides for the flexibility needed for mixed-initiative interaction.

Future Work In this paper, we have focused on the design of SUPPLE and provided the cooking assistant case study to illustrate the approach. It has shown SUPPLE supports sequence expansion by either agent or user which enables the agent to respond to earlier moves in a sequence regardless of the depth of the expansion. In principle, a SUPPLE conversational agent can handle any conversational pattern, but it is of course limited by the patterns that are available to it. To scale up, we will develop an approach to automatically analyze how conversational sequences follow, expand and close one another in real-world conversation data. The recent study by [30] provides a good basis to this end.

Even though our approach facilitates identification of out-of-context dialog moves (e.g., Fig 1.7), it raises additional questions of how to manage this flexibility and how to provide conversational designers with the tools to restrict the scope of application of sequence expansions. Future work should clarify how we can specify such scoping constraints and how the agent should handle dialog moves of a user that conflict with these constraints. Foremost, the case-study is only a proof of concept and usability studies are a crucial next step to empirically assess the usefulness of automatically generating a tree-like chain of predefined patterns based on user and agent dialog moves.

References

[1] Core MG, Allen J. Coding dialogs with the DAMSL annotation scheme. In: AAAI fall symposium on communicative action in humans and machines. vol. 56. Boston, MA; 1997. p. 28-35.

[2] Traum DR, Larsson S. In: van Kuppevelt J, Smith RW, editors. The Information State Approach to Dialogue Management. Dordrecht: Springer Netherlands; 2003. p. 325-53. Available from: https://doi.org/10.1007/978-94-010-0019-2_15.

[3] Henderson M, Thomson B, Young S. Deep neural network approach for the dialog state tracking challenge. In: Proceedings of the SIGDIAL 2013 Conference; 2013. p. 467-71.

[4] Qun H, Wenjing L, Zhangli C. B&Anet: Combining bidirectional LSTM and self-attention for end-to-end learning of task-oriented dialogue system. Speech Communication. 2020;125:15-23.

[5] Moore RJ, Arar R. Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework. Morgan & Claypool; 2019.

[6] De Jaegher H, Di Paolo E, Gallagher S. Can social interaction constitute social cognition? Trends in cognitive sciences. 2010;14(10):441-7.

[7] Blaylock N. Towards Flexible, Domain-Independent Dialogue Management using Collaborative Problem Solving. In: Decalog 2007: Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue; 2007. p. 91–98.

[8] Rosset S, Bennacef S, Lamel L. Design strategies for spoken language dialog systems. In: Sixth European Conference on Speech Communication and Technology; 1999. .

[9] Buck JW, Perugini S, Nguyen TV. Natural Language, Mixed-Initiative Personal Assistant Agents. In: Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication. IMCOM '18. New York, NY, USA: Association for Computing Machinery; 2018. Available from: https://doi.org/10.1145/3164541.3164609.

[10] Allen JE, Guinn CI, Horvtz E. Mixed-initiative interaction. IEEE Intelligent Systems and their Applications. 1999;14(5):14-23.

[11] Hochberg J, Kambhatla N, Roukos S. A flexible framework for developing mixed-initiative dialog systems. In: Proceedings of the Third SIGdial Workshop on Discourse and Dialogue; 2002. p. 60-3.

[12] Jordan P, Ringenberg M, Hall B. Rapidly developing dialogue systems that support learning studies. In: Proceedings of ITS06 Workshop on Teaching with Robots, Agents, and NLP; 2006. p. 1-8.

[13] Perugini S, Buck JW. A Language-Based Model for Specifying and Staging Mixed-Initiative Dialogs. In: Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS '16. New York, NY, USA: Association for Computing Machinery; 2016. p. 204–216. Available from: https://doi.org/10.1145/2933242.2933262.

[14] Allen JF, Schubert LK, Ferguson G, Heeman P, Hwang CH, Kato T, et al. The TRAINS project: A case study in building a conversational planning agent. Journal of Experimental & Theoretical Artificial Intelligence. 1995;7(1):7-48.

[15] Rich C, Sidner CL, Lesh N. Collagen: Applying collaborative discourse theory to human-computer interaction. AI magazine. 2001;22(4):15-5.

[16] Görz G, Ludwig B. Speech Dialogue Systems-A Pragmatics-Guided Approach to Rational Interaction. KI. 2005;19(3):5.

[17] Bohus D, Rudnicky AI. The RavenClaw dialog management framework: Architecture and systems. Computer Speech Language. 2009;23(3):332 361. Available from: http://www.sciencedirect.com/science/article/pii/S0885230808000545.

[18] Wong W, Cavedon L, Thangarajah J, Padgham L. Flexible conversation management using a bdi agent approach. In: International Conference on Intelligent Virtual Agents. Springer; 2012. p. 464-70.

[19] Morbini F, DeVault D, Sagae K, Gerten J, Nazarian A, Traum D. FLoReS: A Forward Looking, Reward Seeking, Dialogue Manager. In: Mariani J, Rosset S, Garnier-Rizet M, Devillers L, editors. Natural Interaction with Robots, Knowbots and Smartphones. New York, NY: Springer New York; 2014. p. 313-25.

[20] Poesio M, Cooper R, Larsson S, Matheson C, Traum D. Annotating conversations for information state update. In: Proceedings of Amstelogue 99, 3rd Workshop on the Semantics and Pragmatics of Dialogues; 1999. .

[21] Ljunglöf P. Dialogue management as interactive tree building. In: Workshop on the Semantics and Pragmatics of Dialogue; 2009. .

[22] Sacks H. Notes on methodology. Structures of social action: Studies in conversation analysis. 1984;21:27.

[23] Erickson T. Lingua Francas for design: sacred places and pattern languages. In: Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques; 2000. p. 357-68.

[24] Levinson SC. Pragmatics. Cambridge University Press; 1983.

[25] Hindriks KV. In: El Fallah Seghrouchni A, Dix J, Dastani M, Bordini RH, editors. Programming Rational Agents in GOAL. Boston, MA: Springer US; 2009. p. 119-57. Available from: https://doi.org/10.1007/978-0-387-89299-3_4.

[26] Sterling L, Shapiro EY. The art of Prolog: advanced programming techniques. MIT press; 1994.

[27] Allen J, Ferguson G, Stent A. An Architecture for More Realistic Conversational Systems. In: Proceedings of the 6th International Conference on Intelligent User Interfaces. IUI '01. New York, NY, USA: Association for Computing Machinery; 2001. p. 1–8. Available from: https://doi.org/10.1145/359784.359822.

[28] Bocklisch T, Faulkner J, Pawlowski N, Nichol A. Rasa: Open source language understanding and dialogue management. 31st Conference on Neural Information Processing Systems (NIPS 2017). 2017.

[29] Nguyen A, Wobcke W. An adaptive plan-based dialogue agent: integrating learning into a BDI architecture. In: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems; 2006. p. 786-8.

[30] Tewari M. Beyond adjacency pairs: Hierarchical clustering of long sequences for human-machine dialogues. In: Computational Approaches to Discourse (CODI), held in conjunction with Empirical Methods in Natural language processing (EMNLP), Virtual meeting, November 16-20, 2020; 2020. p. 11-9.