ARIA Valuspa

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

Artificial Retrieval of Information Assistants – Virtual Agents with Linguistic Understanding, Social skills, and Personalised Aspects

Collaborative Project

Start date of project: **01/01/2015** Duration: **36 months**

(D3.3). Release of the multilingual, adaptive dialogue toolbox

Due date of deliverable: Month 31 Actual submission date: 28/09/2017

ARIA Valuspa

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

Project co-funded by the European Commission		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

STATUS: [FINAL]

Deliverable Nature		
R	Report	
P	Prototype	
D	Demonstrator	**X**
O	Other	

Participant Number	**Participant organization name**	**Participant org. short name**	**Country**
Coordinator			
1	University of Nottingham, Mixed Reality/Computer Vision Lab, School of Computer Science	UN	U.K.
Other Beneficiaries			
2	Imperial College of Science, Technology and Medicine	IC	U.K.
3	Centre National de la Recherche Scientifique, Télécom ParisTech	CNRS-PT	France
4	Universitat Augsburg	UA	Germany
5	Universiteit Twente	UT	The Netherlands
6	Cereproc LTD	CEREPROC	U.K.
7	La Cantoche Production SA	CANTOCHE	France

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

Table of Contents

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

1. PURPOSE OF DOCUMENT

This is the last deliverable for Work Package 3: "*Implementation of adaptive task-based dialogue system*". The leader of this work package is UT, with involvement of the following partners: UON, CNRS, ICL, UA, CNT and CRPC.

The objectives of this work package are two-fold. The first is to build adaptive and task-oriented dialogues in multiple languages to assist information retrieval in general, and for the two scenarios (the book ARIA and the Industry ARIA) in particular. The second objective is to build a framework for information retrieval style dialogue management specification that can be used outside the ARIA-VALUSPA project for adaptive dialogue construction.

This document describes the last version of the dialogue manager, DM2.0, and the steps taken towards this version of the dialogue manager. The DM2.0 is the core component in the (multilingual) adaptive dialogue toolbox that will be released at the end of the ARIA project as part of the ARIA-VALUSPA Platform (AVP). We discuss the concepts and workings of the toolbox.

The deliverable is structured as follows. Section 2 of this deliverable describes DM2.0 in detail. Section 3 describes the progress on all tasks of this work package that were not covered in Section 2. In Section 4 the main conclusions are presented and plans for the remainder of the project are outlined. Section 5 describes the outputs of the past period.

2. DIALOGUE MANAGEMENT

In this section, we describe the new version of dialogue management developed in the last period of the project (DM2.0). Within dialogue management we distinguish three concepts:

- Dialogue Manager (Management templates)
- Dialogues (i.e. scenarios consisting of Move templates)
- Dialogue Engine (Flipper 2.0)

The **Dialogue Manager**, described in detail in Section 2.1, deals with how the agent behaves in an interaction. The Dialogue Manager consists of Management Templates that, for example, define the organisation of information in the information state. See Section 2.2.1 for example management templates.

Dialogues are used to describe the scenario that the agent knows about and can converse about. They are defined in a Dialogue Structure consisting of Move templates, described in detail in Section 2.1.1. Section 2.2.2 shows example Move templates.

The **Dialogue Engine**, described in detail in Section 2.3, consists of the new version of Flipper. Flipper checks the dialogue templates that are used to define the Dialogue Management in the ARIA agent.

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

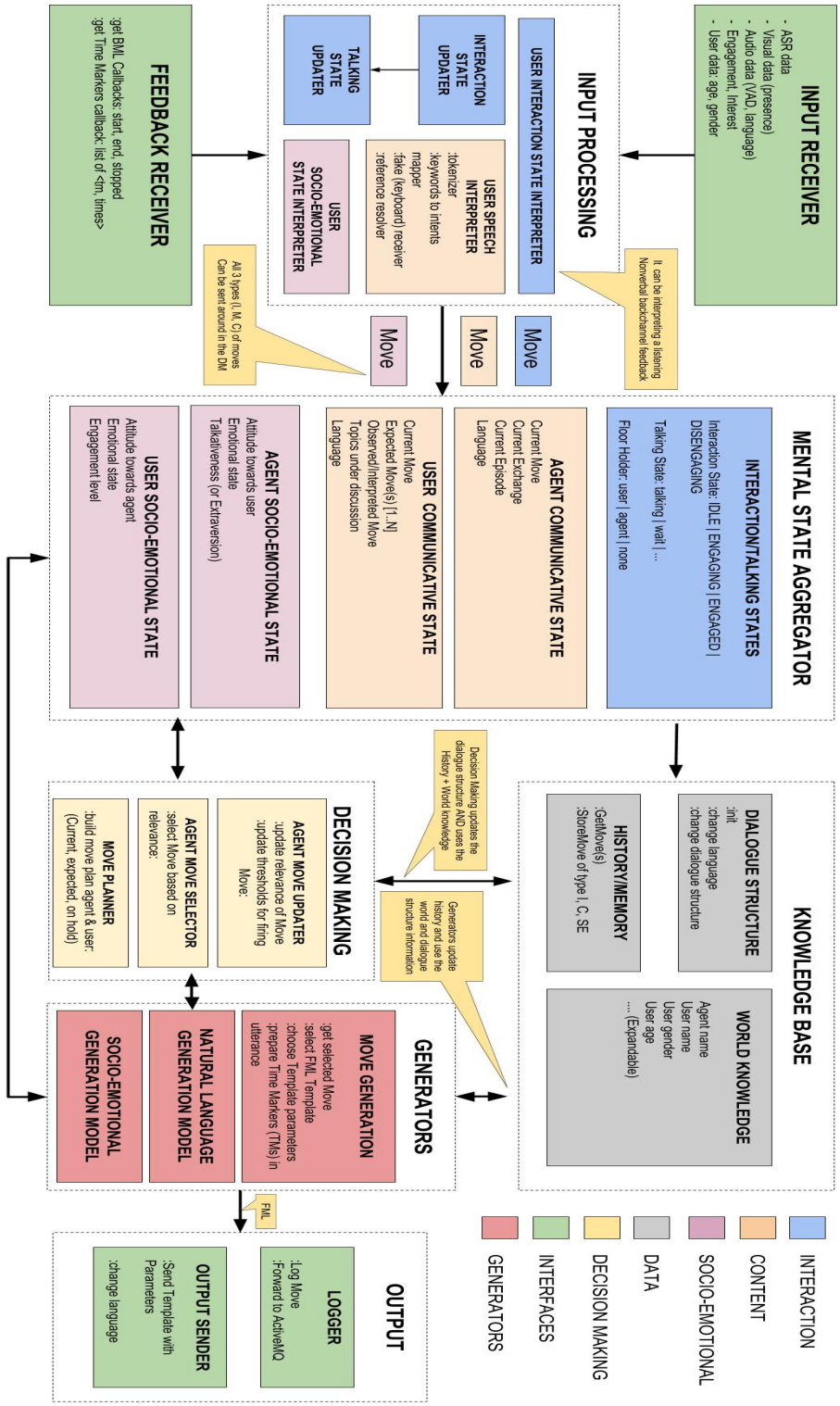

Figure 1: DM 2.0 Structure

2.1 Dialogue Manager 2.0

We have designed a new version of the Dialogue Manager (DM 2.0) that takes a scenario and situation-driven approach to creating dialogue structures based on conversational acts. It shares some properties with current tools for the development of dialogues, such as the use of dialogue trees from DISCO (Rich and Sidner, 2012) and the use of question-answer matching for information retrieval from the NPC Editor of the Virtual Human Toolkit (Leuski and Traum, 2011). Similar to the FLoReS dialogue manager of Morbini et al. (2014), the DM 2.0 has been set up to facilitate the creation of structured dialogues with the use of domain experts.

Below we first describe the way the information in the Dialogue Manager is structured (the Dialogue Structure, Section 2.1.1) and what information is present in the operational system (Section 2.1.2). After this, we describe how the decision to perform specific agent behaviour is made (Section 2.1.3) and how intent planning is done (Section 2.1.4). The conceptual components of the Dialogue Manager and the information flow between them are depicted in Figure 1.

The model has been developed and implemented in Flipper 2.0 (see Section 2.3), and currently a few example dialogues have been authored. DM2.0 will be used as the basis for the development of the industry ARIA (Section 3.7).

2.1.1 Dialogue Structure

Dialogues are defined in terms of hierarchical dialogue acts in a Dialogue Structure, as shown in Figure 2:

- Dialogue structure: this is the root. The name should refer to the name of the character.
- Episode: the first tier consists of episodes. An episode covers a phase (or a very general topic) of a conversation (e.g. social, Q&A, reading along with the agent, unexpected situations)
- Episode.Exchanges: episodes are made up of exchanges that are related to a specific topic in the episode (e.g. two or more utterances about the same topic)
- Episode.Exchanges.Moves: the lowest tier consists of moves. An exchange is made up of several moves. These correspond to conversational acts, based on the DIT++ standard (Bunt et al., 2012). A move can be realized with an utterance, nonverbal behaviour, or a combination of the two. A move has a goal that can be achieved by the behaviour and a status for that goal. Moves are selected for execution based on how *relevant* they are to the situation in the conversation. Rules define when a move becomes relevant.

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

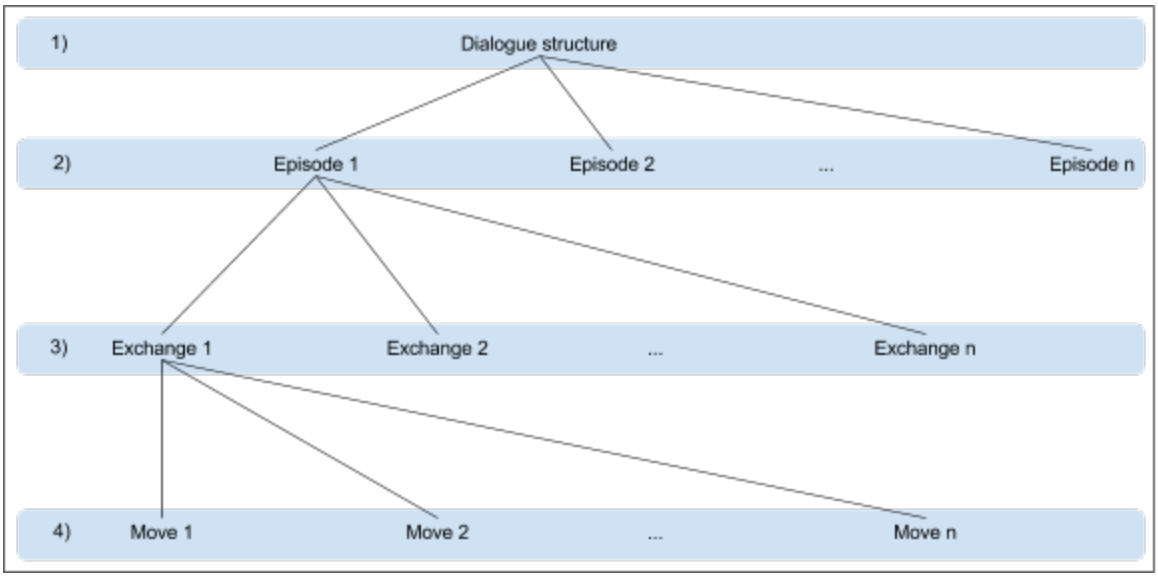

Figure 2: The hierarchical dialogue structure.

Moves are the atomic unit (dialogue item) in the Dialogue Manager (DM). A move can refer to a move (a dialogue act) of the user or the agent. A move can be listening or speaking, depending to its DIT++ category. We distinguish three types of moves, depending on the information they carry:

- Content/Dialogue Act (C): C-Moves deal with information exchange. They contain the content of what the agent or user said, for example if the user asked the question "what can you tell me about Alice in Wonderland?".
- Interaction Management (I): I-Moves contain information that is about the interaction, for example turn-management or establishing contact between the user and agent. They correspond to the Interaction Management functions of the DIT++ taxonomy of dialogue acts.
- Socio Emotional (S): S-Moves express the social and emotional state of the agent or user, for example an increase in dominance or a decrease in sentiment. In DIT++ these are seen as qualifiers of dialogue acts; however we see them as moves since they can have specific (often nonverbal) behaviours associated with them.

Additionally, the content moves *for the agent* are further divided depending on the type of content that is contained within the move. This division helps with planning the agent moves (for example combining content with an opinion) and helps a dialogue scenario author to keep track of what purpose a move has:

- Content (C-tag): This is the type of C-Move that contains information from the book, for example what events happened and who Alice has met. An example would be: "When I saw the White Rabbit, I chased him into a rabbit hole."
- Opinion (O-tag): A C-Move with an opinion tag contains an opinion, for example Alice's opinion about events or characters from the book, such as: "I thought it was rather strange to see a white rabbit with a watch."
- Meta information (M-tag): A C-Move with a meta information tag contains information on a meta level about the interaction. Using such moves, the agent can talk about the interaction, for example during a lull in the conversation the agent might say "Do you want to continue talking about this?".

The DM has been designed with authoring in mind. An author can create dialogues (i.e. episodes, exchanges, moves of the agent) without needing expert knowledge about dialogue systems. In general, an author only needs to specify a situation (e.g. a new unknown user appears, smiles, and says "Hello") and what an agent should do in that situation (e.g. smile and say "Hello, nice to meet you!").

A move has different attributes, some of which need to be filled in by an author, others can be computed automatically. The following attributes need to be authored:[1]

- Name: the path in the dialogue structure (e.g. kb1.episode1.exchange1.move1). We suggest basing the name on the goal of the dialogue item.
- AgentGoal: [DIT++].exchange name (e.g. info_request.QATopic)
- AgentBehaviour: verbal or non-verbal behaviour.
- ContentType: meta | content | opinion
- PreconditionRules (specific):
    - Preconditions that control dialogue flow. For example, after greeting the user, it makes sense for the agent to suggest to the user an action to take or to introduce a topic.
    - Specific User Utterances to which the move would be an appropriate response. For example, the user utterance "how are you doing" is a good indicator that a move that has AgentBehaviour "I'm doing great, thank you for asking" is relevant.

A move has the following attributes that can be computed automatically (in most cases) before an interaction. They can also be authored if needed, or computed during an interaction:

[1] Some of these attributes apply only to C-moves.

- *GoalStatus*: This refers to whether or not the goal of the move has been completed or accomplished, and is updated during the interaction for every move.
- *RefersTo*: The owner (agent | user) of the move
- *Relevance*: A value [0..1] calculated during the interaction.
- *AgentForwardGoal: [DIT++].exchange name (e.g. attention.QATopic).* This is the agent's planned dialogue act that is used to plan the next moves and can be automatically determined using adjacency pairs and floor management or turn-taking rules. For example, for a move with as goal an information_request (i.e. a question), the forward goal of the agent would be attention (i.e. listen to the answer).
- *UserExpectedGoal: [DIT++].exchange name (e.g. i.QATopic)*. This is what the agent the user's **current** dialogue act will be and can be used to make a dialogue plan. For example, when the agent is talking, we expect the user to be quiet.
- *UserForwardExpectedGoal: [DIT++].exchange name (e.g. inform_answer.QATopic).* This goal is what the agent thinks the user's **next** dialogue act will be. This can be used to create a dialogue plan. For example, after the agent has asked a question, we expect the user to start talking.
- PreconditionRules (general):
    - General preconditions (rules that apply to most moves): For example, it doesn't make sense for the agent to talk when no user is present.
    - Situation Specific preconditions (rules that apply to some moves or to moves in some case). For example, it doesn't make sense for the agent to speak right after it has asked a question.

An example of a Move Template, with explanation, can be found in Section 2.3.2.

2.1.2 Operational Dialogue Manager

The Dialogue Engine (Flipper) stores all information the agent knows in the information state. Information comes from various sources and is represented in the form of Moves. During an interaction, the *moves of the user* are created by the system via the Input Processing component (see Figure 1). Some examples of user moves are:

- The user has started speaking, detected by the Voice Activity Detection: an **I-type user move** is generated stating that the user has started speaking (and possibly that this is an interruption when the agent is also speaking).
- The user has spoken, and the automatic speech recognition (ASR) outputs a word string: a **C-type user move** is generated holding the ASR output.
- The user has started smiling, the SSI updates the valence of the user's affective state: an **S-type** user move is generated holding the valence.

Additionally, information about the *agent's actions* is received as input (i.e. feedback) from the behaviour realiser. The behaviour realiser (e.g. Greta) sends continuous feedback about what behaviour has been carried out. Feedback can be:

- BML Callbacks: the BML realiser sends information about which behaviour (BML block) has started, ended, or has been stopped.
- Time Markers Callbacks: during the agent behaviour, the realiser sends feedback on the exact timing of each behaviour that is executed. This is done using time-markers (see Section 2.2.4). For agent utterances, this is done on word level.

The feedback allows the Input Processing to keep track of the floor (i.e. turn-taking) and the completion of the goals of the agent. For example, knowing from the feedback when the agent has stopped speaking and knowing from the ASR when the user has started speaking allows us to determine whether there is overlapping speech and thus whether the user interrupted the agent. Additionally, the time markers allow us to know what part of the agent utterance has been said uninterrupted (and thus was heard by the user) and what part was not heard because it was interrupted. The agent might concatenate moves, for example a C-tagged C-move ("The White Rabbit had a watch") with an O-tagged C-move ("I liked that watch"). Time marker feedback is used to determine whether the goal of the agent's move was accomplished: if the agent was interrupted before it could complete the utterance, the goal of the move (e.g. of conveying this information) is not accomplished. This might mean that the agent will repeat itself.

2.1.3 Decision Making

Moves have rules that determine when the move becomes relevant. The agent selects its moves based on their relevance. We view relevance as the utility value of a move, where the agent is trying to maximize the utilities of moves and selects the moves with the highest relevance above a certain threshold. This threshold is dynamic and decreases when for a certain amount of time no move has been performed and increases when the agent is speaking.

The relevance of a move gets updated by the Agent Move Updater. Relevance is based on the rules in the move. When a rule (defined in the preconditions of the Flipper template) is met, the relevance of the move increases. Additionally, the relevance of a move goes up when the user utterance matches one of the utterances to which this move would be an appropriate response, as predefined in the move. This is an extension of the QA (question-answer) Matcher introduced in the previously delivered version of the ARIA software (see also Section 3.6). Furthermore, relevance of a move increases if closely related moves (e.g. moves in the same exchange) become more relevant. We use Management templates in Flipper to update the relevance of the dialogue structure.

The Agent Move Selector keeps track of the relevance of all the moves in the Dialogue Structure. It selects the move with the highest relevance above the threshold and sends this move to the Move Planner. Additionally, it sends the selected move to the Move Generator (see Figure 1) for execution by the agent embodiment (see Section 2.2.4).

The Move Planner keeps track of the current agent move and the planned agent move. It gets information from the move selector and the Input Processing modules for observed and predicted user moves. The agent keeps track of five moves at any time for the decision-making process:

- The **current agent move**. This is the move which the agent is currently carrying out, for example asking a question or showing attentive listening behaviour.
- The **planned agent move**. This represents the agent's forward-looking move. The agent has a prediction of what its next move may be and puts this move in its behaviour plan. This is planned based on the current agent move, using adjacency pairs. For example, the agent will most likely perform listening behaviour after it has asked the user a question.
- The **expected user move**. Based on the current agent move, we expect the user to behave in some way during the agent's move. Expectations can be based, among other things, on turn-taking rules (e.g. one speaker at a time). For example, when we are talking we expect the user to listen.
- The **observed user move**. This is created via the Input Processing module. For example, if in the previous turn the agent has asked a question and voice activity is detected, it is most likely that the user is now informing us of the answer.
- The **expected user's next move**. This expectation of what the user will do next is created based on the agent's current move and the observed user move. An example is when the agent is asking a question, observes that the user is paying attention, and expects the user to give an answer in the next turn.

The Move Planner checks whether the plan is still correct: are the expected and observed user move the same? When this is not the case, a re-plan is needed and the Move Planner requests the Agent Move Updater to re-evaluate all moves with the new situation.

Summarizing, deciding what move of the agent should do is done by three components, see Figure 1:

- The **Agent Move Updater** uses all the available information in the information state to update the relevance of each agent move in the dialogue scenario (see Section 2.2.1) when the Move Planner calls for an update.
- The **Agent Move Selector** selects the most relevant move, after the Agent Moves have been updated and one or more moves exceed the relevance threshold. It

sends the most relevant move to the Move Planner, and prepares for its execution (see Section 2.2.4).

- The **Move Planner** keeps a plan of the moves that have been selected for execution by the agent and determines which moves are expected from the user. When the plan is finished, or the expected user move is not observed, the Move Planner calls for an Agent Move Update (which constitutes a re-plan).

2.1.4 Intent Planning for Agent Behaviour Generation

Once an agent move has been selected and put in the Move Planner, the Move Generator translates this move to FML-APML. First, the agent's verbal utterance (if present) is extracted from the selected move, and time markers are added to it. Secondly, the emotion of the agent is set, based on the current emotional state of the agent stored in the information state. Furthermore, additional parameters (e.g. backchannel, stance) in the move can be used to fill the placeholders in the FML-templates. Finally, we have implemented a natural language generation module that can adapt the verbal content of an agent move to align it to the user's word choice. The module takes the dialogue history into account and tries to align the agent's utterance to the user's utterances. More details on this module can be found in Section 3.3.

2.2 Examples of templates

2.2.1 Example Management Templates

Here we show two examples of management templates in Flipper 2.0. The first example is a template that passes a user utterance to the QA matching module, when the user has made a content move:

```
<template id="001" name="answerQuestion>
      <preconditions>
            <condition>is.states.user.lastMove.type == c </condition>
      </preconditions>
      <effects>
            <method persistent="is.states.modules.QAMatcher"
is="is.states.agent.say.answer" class="eu.aria.dm.modules.QAMatcher"
name="findBestAnswer">
                  <arguments>
                        <value class="String"
is="is.states.user.say.utterance"
                  </arguments>
            </method>
      </effects>
</template>
```

The second example is a template that makes the agent listen after it has asked a question. This template is part of a larger collection of interaction state templates that indicate when the agent should perform a listening or talking behaviour:

```
<template id="101" name="updateInteractionState>
      <preconditions>
            <condition>is.states.agent.say.askedQuestion</condition>
      </preconditions>
      <effects>
            <assign is="is.states.agent.interactionState">listen</assign>
      </effects>
</template>
```

2.2.2 Example Move Templates

Below we show an example of a possible dialogue move by the agent. In this instance, the agent makes suggestions for a topic.

```
{"Episode":
  "Goal": "Social",
  {"Exchange":
        "Goal": "TopicManagement"
        {"Move":
              "Goal" : "Agent_topic_suggestions",
              "UU": [
                    "What can you do?",
                    "What can you tell me?",
                    "What else can you tell me?",
                    "Do you know other things?"
              ],
              "Rules": "silence > 5 ||
                          QA.$prev_exchange == completed &&
                          prev_episode == QA",
              "AB": [
                    "Well, I'd like to talk about $!prev_topic. OK?",
                    "What do you think about $!prev_topic?",
                    "Do you know $!prev_topic?"
              ],
              "Type": "M"
}}}
```

This move is part of episode "Social", and part of exchange "TopicManagement". The move is called Social.TopicManagement.Agent_topic_suggestions (which is a concatenation of the goals of the hierarchy). UU refers to User Utterances that would

make this move relevant (i.e. to which it would be an appropriate response). The Rules refer to situations in which this move is relevant. In this case, either there has been a silence that lasted longer than 5 seconds, or the previous exchange was part of the episode "QA" and it has been exhausted (i.e. all moves in this exchange have been executed). AB refers to the AgentBehaviour that would be appropriate behaviour in the given context. The Type refers to what sort of content this move has, in this case M (Meta Information): this move talks about the conversation on a meta level, in this case determining what to talk about.

Another type of episode we have designed, and which can be reused in many different dialogue scenarios, is called "SafetyZone". In this episode, the agent tries to restore the conversation after it has broken down, for example because the agent does not understand the user or because no user activity is detected. The agent will stay `safe' in this episode until it discovers a rule that moves it to another episode. The SafetyZone is designed to keep the user engaged and to prevent the agent from making repeated statements such as "I'm sorry, I don't know" or "I'm sorry, I do not understand".

An example of a move in this episode is shown below. Whenever the agent does not see the user anymore, or the user has been looking away for some time, the agent will try to re-establish contact.

```
{"Episode":
   "Goal": "SafetyZone",
   {
    "Exchange":
      "Goal" : "Contact"
         {
          "Move" :
              "Goal" : "reestablish_contact",
              "UU" : [],
              "Rules" : [ "face_presence = false",
                                          "agent_turn = true",
                                          "engagement = true",
                                          "history=true"
              ],
              "AB": "Hello, still there?"
         }
    }
}
```

2.3 Dialogue Engine: Flipper 2.0

To check the dialogue templates that define the Dialogue Management in the ARIA agent we make use of Flipper (ter Maat and Heylen, 2011; see also D3.2). Flipper is an information-state based dialogue engine that updates the dialogue context and decides on which action the agent should take, similar to the implementation of FLoReS (Morbini et al (2014). In the past months, we have developed a new version of the dialogue engine (Flipper 2.0). This new version improves the original in several respects. To make it more robust, versatile and easy to use, the following new features have been added in Flipper 2.0:

- Standardization (JSON, JavaScript)
- Optimization
- Usability and error handling
- Features
- Persistency

2.3.1 Standardization

We have changed the representation of the information state from a custom designed Java-model to JSON, hence making it easier for different modules of the Dialogue Manager to retrieve information from the information state.

Furthermore, we have replaced Flipper's old system for evaluating template preconditions by JavaScript evaluation, making it possible for users to use both self-written JavaScript or load JavaScript libraries. We kept the XML format as similar as possible to the previous Flipper, because we found that XML is the best readable format for non-technical people who still want to write custom templates.

Finally, input modules send user input in standardized formats (XML or JSON). This makes it easier to put the user input into the information state.

2.3.2 Optimization

The previous version of Flipper made very frequent use of parsing and evaluation but did not use many standard libraries for this. We have upgraded this to run more efficiently (lazy evaluation and use of standard libraries for parsing XML and JSON). This reduced the code by at least 30% and made it also more readable for maintenance.

2.3.3 Usability and error handling

The templates of the first version of Flipper can still be used in the new version, as the versions are backwards compatible. The components used by a particular template (e.g. parts of the information state or functions called) are always defined or referred to in the templates, so that the author only needs to check a particular template.He does not

have to search through all the code and templates to find out what the impact is of changing said template.

Error handling has been implemented with precise notifications if something went wrong, making it easier for other developers to find errors in their connected Java-modules or XML-templates. Also, it is possible to retrieve certain system parameters for checking which template has been used, which is useful for testing purposes.

2.3.4 Features

In the first version of Flipper, it was not easy to use external modules (e.g. natural language understanding and generation) and all reasoning and logic for the agent was in the templates. We can now create templates that can use external modules to do more complex calculations without cluttering the templates, which should only describe the dialogue structure on a higher level.

As mentioned before, JavaScript is now a part of Flipper 2.0. This feature improves the flexibility of defining preconditions and manipulating the information state. Furthermore, existing JavaScript libraries can be added easily.

2.3.5 Persistency

In Flipper 2.0, the information state can only be changed from within Flipper templates, no longer from outside classes. A database connection has been made possible with PostgreSQL. We commit and store the information state in this database, so that the information state can be retrieved or checked when it has changed (e.g. for checking logs of agent behaviours) or be restored when data becomes corrupted (e.g. when the Internet connection fails). This is called persistency. It makes the system more robust to unwanted behaviour.

Because all the information that is in the information state at some moment during an interaction is kept in the database, we could use it to restore the dialogue to a previous state, for example, the state when the current user had a previous conversation with the system. This makes it possible for the agent to refer to this previous conversation or for the user and agent to continue their conversation where they left off. (Implementing this type of behaviour is future work.)

Furthermore, we can store various kinds of background knowledge in the database. We can use it to store all long-term information, compared to short-term information in the information state. For example, the full dialogue history and all previous user demographic information can be stored in the database.

3. PROGRESS PER TASK

In this section, we discuss the progress per task that was not covered earlier in this document.

- Task 3.1, Multi-lingual natural language understanding
- Task 3.2, Task-oriented dialogue management
- Task 3.3, User-adaptive dialogue strategies
- Task 3.4, Reinforcement learning based on user feedback
- Task 3.5, Dealing with unexpected situations
- Task 3.6, Generation of Dialogues for Book Personification demonstrator
- Task 3.7, Generation of Dialogues for Industry Associate demonstrator

3.1 MULTI-LINGUAL NATURAL LANGUAGE UNDERSTANDING

This task concerns the extension of the ARIA-VALUSPA agent's shallow natural language understanding skills to handle the three natural languages targeted by the project: English, French and German.

Three different Automated Speech Recognition (ASR) modules have been developed to recognize speech in English, German and French. Recognized word strings are put in the information state, and the Input Processing component of the dialogue system is informed of the new user utterance. The Input Processing component then tries to map the user utterance to the user utterances defined in the moves (see UU in the example template from Section 2.3.2), which are currently only in English. To make this multilingual, the utterances of both the agent and user will be translated to the foreign languages.

The mapping of recognized user utterances to user utterances specified in the templates is currently done in a language-independent fashion by measuring unigram overlap between the strings; it does not involve syntactic or semantic parsing and therefore does not require any language-dependent resources. Therefore simply translating the templates is sufficient to have dialogues in French or German in addition to English. Future users of the system can author the dialogue templates in their preferred language, provided that an ASR component and a TTS component are available for that language. For non-verbal behaviour, we make the simplified assumption that there is no difference amongst the different languages.

3.2 TASK-ORIENTED DIALOGUE MANAGEMENT

This task involves the implementation of an information state-based architecture for dialogue management that can run different dialogue management dimensions in parallel, focusing on interaction management aspects such as turn taking, repair mechanisms, and floor management. The work in this task has been extensively covered in Section 2.

3.3 User-adaptive dialogue strategies

A special new feature of the ARIA-VALUSPA dialogue management is adaptation to the user. Adaptation, or alignment, in natural dialogues appears in many aspects of dialogue. This task involves the implementation of different adaptive strategies that are relevant to the application.

One simple form of user adaptation is by making use of the input provided by the SSI framework. The SSI can detect demographics of a user (e.g. the gender) and this can be used by the agent to create an appropriate of responses, for example, "Hello Sir" versus "Hello Madam". Other, more advanced forms of adaptation we have focused on are adaptation in terms of turn-taking and alignment of the agent's word choice to that of the user.

3.3.1. Turn taking

Knowing when the user speaks and adapting the agent's behaviour to this is important for a smooth dialogue. To make this possible, we developed an Interaction State Manager, which is located in the Input Processing component of DM2.0 (see Figure 1). It collects information about a user being present and the current floor and turn-taking situation in the dialogue. This information is passed to the information state, from where it can be used by the rest of the DM. The Interaction State Manager keeps track of the agent's interaction state: if it is idle, trying to engage or disengage, or if it is engaged with a user. Additionally, the agent uses a model of when the agent is talking, listening, interrupting, yielding a turn, or waiting, shown in Figure 3. This supports dynamic turn-taking and can be used to update the relevance of moves as well.

European Union's Horizon 2020 research and innovation programme 645378, ARIA-VALUSPA

September, 2017

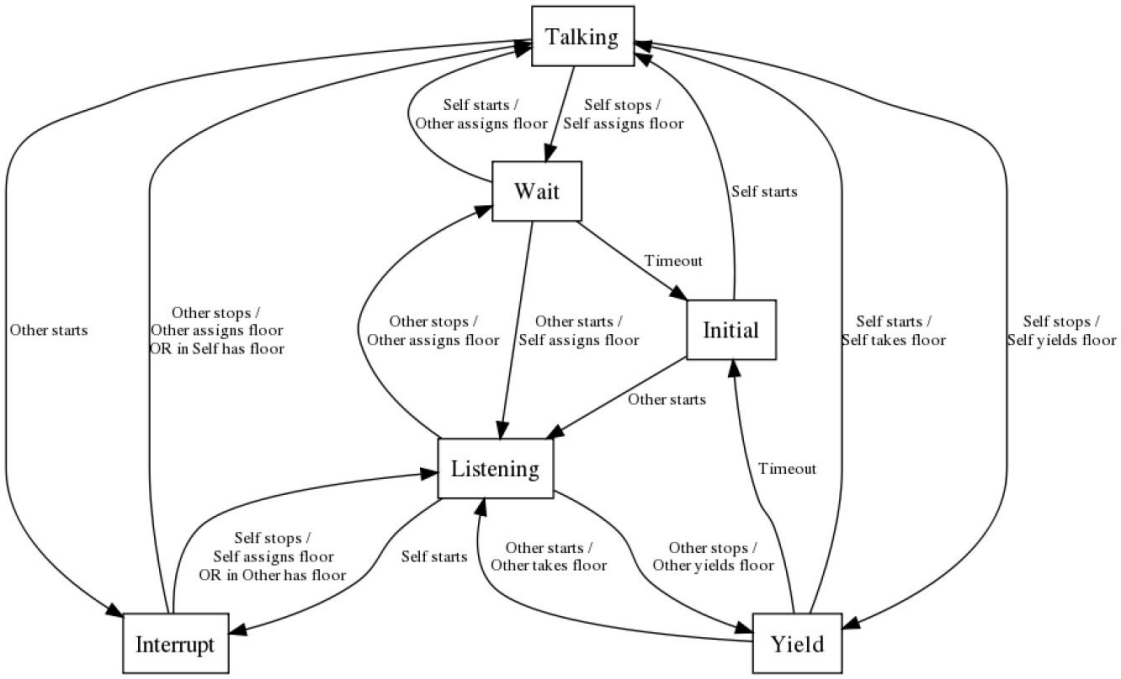

Figure 3: Interaction State Model, based on the work of Loch (2011)

3.3.2 VERBAL ALIGNMENT

In addition to the verbal alignment described in Section 2.3 in D4.3, we have developed rules for adapting agent utterances to those of the user. This makes it possible for the agent to use referring expressions from the book that are preferred by the user. An example is talking about the 'tiny golden key' found by Alice in the book. We could refer to it in the agent utterance as 'key', 'golden key' or 'tiny key'. If the user uses a particular description, the agent is able to mirror this. To achieve the alignment, we adopt a text-to-text generation approach. This means we automatically modify the manually specified agent utterances, instead of generating aligned referring expressions from scratch, as done in the work on alignment of referring expressions by Buschmeier et al. (2009) and Gatt et al. (2011).

First, to determine the user's preferred expression, the user input is stemmed, part-of-speech tagged and divided into chunks. Stemming and chunking were chosen over their more sophisticated counterparts (i.e. lemmatization and full syntactic parsing) to make the whole process more resilient to potential inaccuracies of the ASR. The same processing is done for the sentence defined in the FML, i.e. the agent answer, but in this case we can also obtain a syntactic tree. For stemming, part-of-speech tagging and

parsing the Stanford CoreNLP toolkit is used, while the TextPro[2] suite divides the text into chunks. We then proceed to adjust every noun phrase (NP) in the agent sentence whose head stem matches a noun in the user sentence, adding and removing adjectives and adverbs so that the phrase of the agent matches the chunk of the user. Additional constraints make sure the phrases are not modified when it is inappropriate to do so, e.g. in existential questions (when the user asks, "are there any cats in this story?", the agent should not answer "yes, there are *any* cats") or inside quotations (when the agent is reporting something said by others).

By modifying every NP, we are creating a sentence that -by construction- maximizes the alignment with the user. However, the process also involves a number of intermediate steps where not every noun phrase is aligned (overgeneration). This means that, depending on its goal, the agent can also choose different sentences, e.g. shorter ones to be concise, or those that minimize the alignment to see if the user perceives the agent differently.

We are currently working on extending this process with synonyms and antonyms. For example, if the user references the same object using different synonyms, we might have the agent applying this strategy as well. Finally we are in the process of setting up an experiment to see to what extent the difference between unaligned and aligned dialogues is noticeable by users, and if this contributes positively to the dialogue.

3.4 Reinforcement Learning based on user feedback

This task refers to training the system's adaptive dialogue strategies using reinforcement learning. It was originally planned for the months 31-36, but not updated correctly when the delivery date of D3.3 was moved forward from month 36 to month 31. This task has thus just started to be actively pursued.

3.5 Dealing with unexpected situations

This task concerns enabling the ARIA agents to deal with unexpected situations that occur during an interaction. Two types of unexpected situations that we have addressed involve turn-taking. We have designed a model for dealing with interruptions by the user: the agent can employ re-planning, holding and halt behaviour (see the main description in D4.3). Additionally, the agent can interrupt the user. This is achieved by moves that become relevant while the user is speaking and by using incremental ASR output. Using moves with dynamic relevance updating allows us to interrupt the user at an opportune moment. For example, if the agent wants to disagree with something the user is saying, a move with rules describing exactly this situation will get a high

[2] http://textpro.fbk.eu

relevance. The agent will start talking, even though it has not gotten the turn, thus interrupting the user.

Additionally, we have defined an episode called 'unexpected situations'. This episode contains exchanges and moves dealing specifically with unexpected situations (ironically making them expected). For example, when we observe the user not looking at us while the agent is talking, the agent can respond to this situation with a remark (e.g. "Well, I think it is polite to pay attention…", see Section 2.2).

3.6 Generation of Dialogues for Book Personification Demonstrator

This task is devoted to the generation of dialogues for the book personification application. The aim is to create a set of dialogue structures that covers the themes in the book and avoids open domain conversation. A first version of the book personification demonstrator was described in D3.2. Since then, the demonstrator has been extended with question-answer (QA) matching functionality comparable to that of the VH Toolkit (Hartholt et al, 2013). A question by the user is mapped to the most similar question found in a database with question-answer pairs, and the corresponding answer is returned.

Currently the demonstrator uses a combination of QA matching functionality and DM 1.0 templates (to be replaced with DM 2.0 episodes and moves as described in Section 2.1). The DM2.0 version of the demonstrator will be delivered later this year.

3.6.1 Question generation.

To easily expand the range of user questions that the Book Personification agent can answer through QA matching, we have developed a question generation system that takes text as input and generates a large number of QA pairs from it. This form of question generation can be used to complement or replace data collection with human users, which we have used in the past to populate the QA database (see D3.2). For example, based on the input sentence (from a summary of Alice's Adventures in Wonderland) "Her giant tears form a pool at her feet" the following questions can be generated: "What happens at her feet?" Or: "What happens to her giant tears at her feet?" The answer to both questions is "Her giant tears form a pool".

Most question generation systems are used in educational applications, such as skill development assessment and knowledge assessment. Applications in conversational characters are rare. One of the exceptions is the work of Yao et al. (2012). Their question generation approach is based on syntactic transformation from declarative sentences into questions. As source texts for question generation they used Wikipedia. In contrast, our approach is based on semantic information (semantic role labels augmented with dependency structures) and we use narrative texts as a source.

So far, we have been using a small corpus of online summaries of *Alice's Adventures in Wonderland* as source texts for question generation, but note that our approach is sufficiently general to take all kinds of texts as source material. We have used summaries instead of the book itself, to avoid potential problems with character dialogue in the form of direct speech and the narrative style used in the book.

Our Question Generation system has been inspired by the work of Mazidi and Nielsen (2014; 2015). We use two types of information about a sentence as a basis for question generation: (1) the semantic role labels assigned to the sentence and (2) the dependency structure of the sentence. Semantic role labeling parses a sentence into a predicate-argument structure with consistent argument labels. Following Mazidi and Nielsen, we use the tool SENNA (Collobert et al., 2011) to produce the semantic role labels for each input sentence. SENNA produces a separate predicate-argument structure for each clause in the sentence. For determining the sentence's dependency structure, we currently use PyStanfordDependencies,[3] a Python interface for parsing to Stanford Dependencies.

The first step of question generation involves matching the semantic role labels in the input sentence to a set of previously established patterns. For instance, our example sentence above matches the pattern A0-V-A1-LOC, where A0 corresponds to the agent role, A1 the patiens role, and LOC is a locative modifier: *Her giant tears (A0) form (V) a pool (A1) at her feet (LOC)*. In the next step, based on the detected patterns (where necessary combined with dependency information), question and answer pairs are formed from each clause. Currently, we distinguish around 20 different patterns for question generation; these are based on the semantic role combinations that occurred most frequently in our corpus of summaries. These patterns have been refined and improved after tests with new (previously unseen) input data. In general, a QA pair can be created if a sentence has a predicate (V), two or more verb arguments and zero or more modifiers: temporal (TMP), locative (LOC), adverbial (ADV) or Manner (MNR). Sentences with fewer than two basic arguments are currently excluded, because they tend to be relatively uninformative, and thus not a good basis for questions and answers. On average, the question generation system produces 4 QA pairs per sentence in the input document.

3.6.2 Follow-up question strategy

An important challenge with using question generation for conversational agents is that there is a high probability of mismatches between the generated questions and the actual questions that the users ask (Yao et al., 2012). Moreover, an inherent limitation of the question generation approach is that it can only generate questions to which the answers can be found in the source text (the book or a summary of the book). To address these challenges, we developed an innovative dialogue strategy that nudges the

[3] https://github.com/dmcc/PyStanfordDependencies

users towards asking follow-up questions that can be matched to the questions in the QA database.

To increase the likelihood that the user of the ARIA system will actually ask this type of questions, we designed and implemented the following strategy. The system does not just present an answer, based on a specific sentence in the book, but follows it up with a hint about the next sentence, to encourage the user to ask a follow-up question about this. This hint takes the form of the clue "then something happens" plus a phrase from the next sentence (if applicable). The following example is an excerpt from an actual dialogue that took place during our user test, which is described below:

> Agent: She follows the rabbit down the hole . Then something happens when She falls for a long time ...
>
> User: What happens after she falls for a long time?
>
> Agent: She falls for a long time and finds herself in a long hallway full of doors

When the user takes up the hint for the follow-up question and repeats it in full, the user's question can be easily matched with the corresponding question in the QA database, as shown in the example. However, if the user takes the hint but asks an elliptic question ("What happens?") this question matches many different questions in the database. To prevent incorrect matches, the system expands elliptic questions based on the hint it has just given, converting them to full questions that have a direct match in the QA database. The following dialogue excerpt from our user test illustrates this. Here, the user question "What happen there" is converted to "What happens when the caterpillar and Alice get into an argument" and matched correctly to the corresponding answer.

> Agent: She meets a caterpillar sitting on a mushroom and smoking a hookah (i.e. , a water pipe) . Then something happens when The caterpillar and Alice get into an argument ...
>
> User: What happen there?
>
> Agent: The caterpillar and Alice get into an argument but before the Caterpillar crawls away in disgust , he tells Alice that different parts of the mushroom will make her grow and shrink . Then something happens when Alice tastes a part of the mushroom ...

3.6.3 User test and next steps

The follow-up question strategy was implemented and tested in a small-scale user experiment in which four users interacted with a standalone version of the QA matcher, with typed text input and output. The QA database used for the test consisted of 691 QA pairs, generated from three summaries of Alice's Adventures in Wonderland. The dialogues with the test users have not yet been analysed in detail, but the first impression is that the follow-up question strategy works well to give the users some

idea of what to ask, and to suggest questions to them that can actually be answered. However, users also frequently asked questions that had no match in the database (e.g., "Who is the mad hatter" or "What is the King's name?"). To make it possible for the system to answer such questions as well, we need to use additional sources for question generation, e.g., Wikipedia entries about the book and its characters. The question generation approach could be easily expanded to include such sources. In addition, the question generation system needs to be extended to generate multiple variations of the same question, to ensure that a correct match can be found even when the user deviates from the phrasing that was used in the source text and in the system's hints.

The next step will be to expand the QA database used in the Book Personification demonstrator with the already generated QA pairs from the book summaries[4], and with additional QA pairs to be generated from other sources as suggested above. The follow-up question strategy also needs to be added to the Book Personification demonstrator. A link to the software can be found at the bottom of this document.

3.7 Generation of Dialogues for Industry Associate Demonstrator

This task is devoted to the Industry Associate application. It encompasses defining the exact mission of the Virtual Assistant, creating a dialogue scenario to achieve this goal, and building adaptive and task-oriented dialogues in multiple languages.

Due to legal issues between the industry partner and the project, there has not been much progress on this task. These issues have been sorted out only very recently, and we are currently awaiting the industry partner's input on the desired dialogues so that we can start implementing these in the DM2.0.

[4] Before they can be used in the Book Personification demonstrator, the generated questions and answers still need to be converted from third to first and second person respectively, when referring to Alice.

4. Conclusions and Plans for Next Period

4.1 Conclusions

We have developed a new dialogue engine, Flipper 2.0, and designed a new dialogue manager (DM2.0) that we are developing around this system. This dialogue manager provides a dialogue structure based on the DIT++ standard, that makes it possible to specify domain-specific as well as domain-independent dialogue moves that cover multiple conversational dimensions. The full DM 2.0 is not yet available, but is planned to be made available within the next period, including dialogues for both the book-ARIA and industry-ARIA. We will provide an authoring tool for developing dialogues for other domains as well.

In addition to our work on the dialogue manager, we have developed new techniques for the agent's verbal alignment to the user and for populating a QA database for information retrieval with automatically generated question-answer pairs.

4.2 Plans for next period

The most crucial task in the final months in this project will be finishing the DM 2.0 (i.e. implementing the moves, linking the behaviours, finishing the authoring tool) and its documentation. This will allow others to use the software to create their own ARIA agents. We will finish the implementation and definition of dialogue scenarios for the book-ARIA and the industry ARIA. These will be good showcases of the system and its capabilities and they can help others to understand how to create their own ARIAs.

The open-source nature of this project means that interested parties will be allowed to create an ARIA on their own after the project with our software. The delivered software comes with sufficient documentation to allow industry partners (and external parties) to create a compelling conversational agent.

Authoring of dialogues will be easier with the ARIA DM2.0 than it is with other DM systems, because where possible an author only needs to describe a situation and what the agent should do in that context. To make authoring even simpler, we have started with the development of a visual software tool for authoring of moves.

On the part of reinforcement learning we will use the relevance of a move as a basis for learning. We will set up a small experiment in which we show agent-human dialogues (or dialogue fragments) to observers, who have to rate the quality of the agent's responses. These scores will be used to learn a better relevance update policy.

Already clear is that the system will continue to be developed after the project has ended. We will add a topic planner, which will calculate possible topic transitions that the agent can use for talking about topics the agent wants to talk about or topics the user

finds interesting. Currently, topic recognition in dialogues often works on the utterance level (select the noun phrase in the subject or object position), but it could be extended to determine the topic of a dialogue segment (Rats, 1996; Mei, Bansal and Walter, 2017). To learn how an agent should perform segment-level topic recognition, we will set up an experiment in which observers indicate topics in the dialogues of the corpus. With the annotation of the corpus on topic transitions, we will investigate cues for topic transitions and create a topic management module based on our analysis.

Finally, we want to have users talking to a version of Alice that contains topic management and can recognize higher-level topics and using human-like topic transition strategies. To evaluate these dialogues, observers will rate the recorded conversations using metrics such as human-likeness and competence (Glas, 2015).

5. OUTPUTS

The outputs with pertinence to this deliverable that have been published (or are *in press*) or have been otherwise disseminated so far in this project.

Publications:

Bowden, K., Nilsson, T., Spencer, C., Cengiz, K., Ghitulescu, A. and van Waterschoot, J. (2017) *I Probe, Therefore I Am: Designing a Virtual Journalist with Human Emotions.* Proceedings of the 12th Summer Workshop on Multimodal Interfaces (eNTERFACE '16), pp. 47-53, July 18 – August 12, 2016 Enschede, The Netherlands.

Cafaro, A., Bruijnes, M., van Waterschoot, J., Pelachaud, C., Theune, M., & Heylen, D. (2017). *Selecting and Expressing Communicative Functions in a SAIBA-Compliant Agent Framework*. In proceedings of Intelligent Virtual Agents 2017.

Lina Fasya, E. (2017). Automatic question generation for virtual humans. Master thesis, University of Twente, August 2017.

Kolkmeier, J., Bruijnes, M. and Reidsma, D. (2017) *A demonstration of the ASAP Realizer–Unity3D Bridge for Virtual and Mixed Reality Applications*. In proceedings of Intelligent Virtual Agents 2017.

van Waterschoot, J. and Theune, M. (2017) *Topic-Based Personalization of Dialogues with a Virtual Coach*. In proceedings of the workshop on Persuasive Embodied Agents for Behavior Change, at Intelligent Virtual Agents 2017.

Posters:

van Waterschoot, J. (2017) *Topic recognition and management in conversational agents*. Poster presented at the Young Researchers' Roundtable on Spoken Dialog Systems, August 13-14, Saarbrücken, Germany.

Workshops:
Agents in Practice - Designing for Dialogues. SIKS workshop/tutorial. 2017, March

Workshop on Conversational Interruptions in Human-Agent Interactions. At Intelligent Virtual Agents, 2017.

Invited talks:
Heylen, D.K.J. *Invited Talk* at Workshop on Conversational Interruptions in Human-Agent Interactions. At Intelligent Virtual Agents, 2017.

Heylen, D.K.J. *A Sentimental Journal. Emotions in Daily Life*. Invited talk at the 3rd International Workshop on Emotion and Sentiment in Social and Expressive Media (ESSEM 2017): User Engagement and Interaction. At the International Conference on Affective Computing and Intelligent Interaction, 2017.

Software:
Flipper2.0: https://github.com/hmi-utwente/Flipper-2.0

Alice-QuestionAnswer Generation: https://github.com/evania/alice-qg

Full Aria-Valuspa Platform (AVP): https://github.com/ARIA-VALUSPA/AVP

REFERENCES

H. Bunt, J. Alexandersson, J. W. Choe, A. C. Fang, A. C., K. Hasida, V. Petukhova ... and D.R. Traum. ISO 24617-2: A semantically-based standard for dialogue annotation. In *LREC* pp. 430-437, 2012.

H. Buschmeier, K., Bergmann, and S. Kopp. An alignment-capable microplanner for Natural Language Generation. In Proc. 12th European Workshop on Natural Language Generation (ENLG'09) pp. 82–89, 2009.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. Journal of Machine Learning Research, vol. 12, pp. 2493–2537, 2011.

A. Gatt, M. Goudbeek and E. Krahmer. Attribute preference and priming in reference production: Experimental evidence and computational modeling. Proceedings of the Annual Meeting of the Cognitive Science Society, vol. 33, pp. 2627-2632. 2011.

N. Glas. Engagement driven topic selection for an information-giving agent. Workshop on the Semantics and Pragmatics of Dialogue, SemDial 2015, Gothenburg, Sweden.

A. Hartholt, D. Traum, S. C. Marsella, A. Shapiro, G. Stratou, A. Leuski, L.-P. Morency and J. Gratch, All Together Now: Introducing the Virtual Human Toolkit. In International Conference on Intelligent Virtual Humans, Edinburgh, 2013.

A. Leuski and D. Traum. NPCEditor: Creating Virtual Human Dialogue Using Information Retrieval Techniques, In AI Magazine, volume 32, 2011.

F. Loch. Computational Models for Turn Management using Statecharts. Master Thesis, University of Twente, 2011

M. ter Maat and D.K.J. Heylen. Flipper: An Information State Component for Spoken Dialogue Systems. In: Vilhjálmsson H.H., Kopp S., Marsella S., Thórisson K.R. (eds) Intelligent Virtual Agents. IVA 2011. Lecture Notes in Computer Science, vol 6895. Springer, Berlin, Heidelberg, 2011.

K. Mazidi and R. D. Nielsen. Linguistic considerations in automatic question generation. In the Proceedings of ACL 2014, pp. 321–326.

K. Mazidi and R. D. Nielsen. Leveraging multiple views of text for automatic question generation. In: Proceedings of the International Conference on Artificial Intelligence in Education (2015), pp. 257–266.

H. Mei, M. Bansal and M. R. Walter. Coherent Dialogue with Attention-Based Language Models. In: *AAAI*. 2017. p. 3252-3258.

F. Morbini et al. "FLoReS: a forward looking, reward seeking, dialogue manager." *Natural interaction with robots, knowbots and smartphones*. Springer, New York, NY, 2014. pp 313-325.

C. Rich and C. Sidner. Using collaborative discourse theory to partially automate dialogue tree authoring. In *Intelligent Virtual Agents* (pp. 327-340). Springer Berlin/Heidelberg, 2012

X. Yao, E. Tosch, G. Chen, E. Nouri, R. Artstein, A. Leuski, K. Sagae, and D. Traum. Creating conversational characters using question generation tools. Dialogue & Discourse, vol. 3, no. 2, pp. 125–146, 2012.