Hybrid Conversational AI for Intelligent Tutoring Systems

Charuta Pande[a], Hans Friedrich Witschel[a], Andreas Martin[a] and Devid Montecchiari[a,b]

[a] *FHNW University of Applied Sciences and Arts Northwestern Switzerland, School of Business, Riggenbachstrasse 16, 4600, Olten, Switzerland*

[b] *UNICAM University of Camerino, International School of Advanced Studies, Via Andrea D'Accorso, 16, 62032, Camerino (MC), Italy*

Abstract

We present an approach to improve individual and self-regulated learning in group assignments. We focus on supporting individual reflection by providing feedback through a conversational system. Our approach leverages machine learning techniques to recognize concepts in student utterances and combines them with knowledge representation to infer the student's understanding of an assignment's cognitive requirements. The conversational agent conducts end-to-end conversations with the students and prompts them to reflect and improve their understanding of an assignment. The conversational agent not only triggers reflection but also encourages explanations for partial solutions.

Keywords

Conversational AI, Intelligent Tutoring Systems, Problem-based Learning, Project-based Learning

1. Introduction

Group assignments are common practice in higher education because of their high learning efficiency [1] – students can learn from each other, and teachers have less workload in coaching and giving feedback than individual assignments. Besides efficiency, another benefit of group work is students' ability to acquire team-working skills – such as resolving conflicts or giving and receiving help – that they will need in their career. Finally, students working in groups are frequently observed to motivate each other, build on each others' ideas, and gain self-esteem [2].

However, as pointed out by Webb [2], various less desirable practices may arise when students work on group assignments, some of which will result in suboptimal learning progress. Many of these problems depend on how students are evaluated: if the assessment focuses on a group's productivity, i.e., the quality of the outcome, groups often divide the work to be more efficient. Helping each other and sharing knowledge are then considered inefficient. Students who are less

In A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, F. van Harmelen (Eds.), Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021) - Stanford University, Palo Alto, California, USA, March 22-24, 2021.

charuta.pande@fhnw.ch (C. Pande); hansfriedrich.witschel@fhnw.ch (H.F. Witschel); andreas.martin@fhnw.ch (A. Martin); devid.montecchiari@fhnw.ch (D. Montecchiari)

0000-0001-6530-5401 (C. Pande); 0000-0002-8608-9039 (H.F. Witschel); 0000-0002-7909-7663 (A. Martin); 0000-0002-8969-1973 (D. Montecchiari)

CEUR Workshop Proceedings (CEUR-WS.org)

keen to receive an outstanding grade may also rely on their colleagues and contribute little ("free riders" [3]). Consequently, students often acquire only a few and specialized knowledge.

Solutions proposed for this problem range from group formation approaches [3, 2] over hybrid forms of group-individual tasks [4] where students must mark their contributions to other variants of assessment that encourage true teamwork, knowledge sharing, and, above all, taking responsibility for one's learning [2].

Our goal is to introduce a novel approach of fostering individual and self-regulated learning in group assignments. According to English and Kitsantas [5], who study the relationship between problem-/project-based learning (PBL) and self-regulated learning (SRL), as depicted in Figure 1, reflection – as a final stage in PBL – helps students to become aware of new knowledge acquired in the project, or, in some cases, of knowledge gaps. The teacher should actively support reflection processes: there is evidence that, if teachers skip this final phase, students tend to learn less [6].

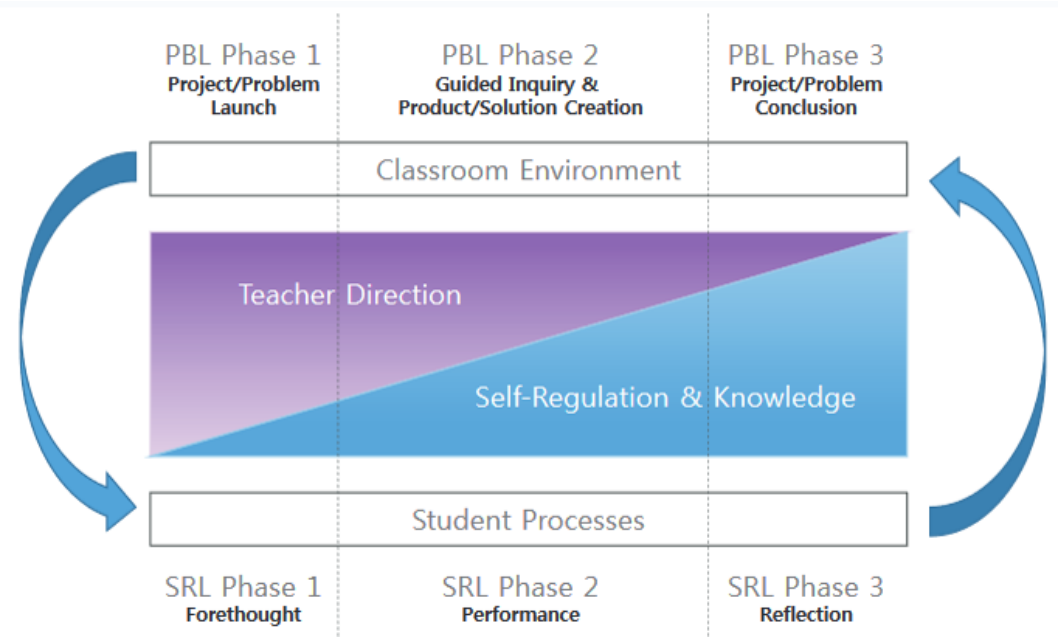

Figure 1: Relationships between PBL and SRL, taken from English and Kitsantas [5]

.

Since there is very clear evidence that individual feedback is superior to group feedback in collaborative learning environments [7], we aim at supporting individual reflection. Because of teachers' limited availability, we want to deliver feedback and questions for reflection in an automated way through a conversational system.

The challenges for constructing such a conversational system are manifold: on the one hand, the system needs to have domain knowledge regarding the assignments that students are solving and knowledge about effective didactic reflection strategies. On the other hand, the system

should react to various ways of how students express themselves and respond with clear and human-like sentences. We believe that the challenges require a combination of explicit knowledge (domain and didactic) with machine-learned, implicit knowledge (language understanding and expressiveness).

Therefore, we will demonstrate in Section 3 how domain knowledge can be elicited from teachers and represented in a tree-like structure that reflects the domain's conceptual relations and, above all, "top-down" solution strategies. Didactic considerations regarding reflection will be incorporated into a dialog management strategy that not only asks how tasks and subtasks were solved but also challenges students to justify their choices, see Section 4. We have implemented the system and will illustrate the ideas with an example dialog. We call our conversational system, the Digital Self-Study Assistant (DSSA). Finally, we present strategies for machine learning-based natural language understanding and generation and how it will be combined with the knowledge-engineered system once enough training data has been acquired (see Section 5).

2. Related Work

In this section, we describe common approaches to design and implement conversational systems, followed by a discussion about conversational agents in education, specifically in the domain of ITS.

2.1. Conversational AI

A conversational system, also referred to as an agent, typically consists of a Natural Language Understanding (NLU) component to interpret the user's utterances, a Dialog Management (DM) component to remember the state of the conversation and identify the next action, and a Natural Language Generation (NLG) component to respond to the user [8, 9].

The literature distinguishes conversational agents as task-oriented, where the dialog system assists users in achieving specific goals [10], or non-task-oriented, which are general-purpose dialog agents and carry out conversations in the form of chitchat [11]. The latter are also called chatbots, although the term may be used broadly to represent a conversational agent in general. Conversational agents in education usually fall into the category of task-oriented dialog agents that are designed around frames. A frame-based dialog system uses slots representing domain-specific information in the form of intents and entities identified from the user's utterances and indicating what a system needs to know [12]. A common and still prevalent approach to slot filling is to use handcrafted rules [12, 9, 13].

Tracking the state of the conversation is also a crucial task in conversational agents, as it determines the next action depending upon the history of the conversation. This functionality is handled in different ways in the literature as well as in commercial conversational agent tools. For instance, Dialogflow [14] uses the concept of *contexts* to remember the state of the conversation and control the flow of the conversation. Rasa Core [15], on the other hand, maintains a stateful *tracker* object for every conversation and stores corresponding slots and events. Approaches for dialog management follow handcrafted rules as well as probabilistic learning [9].

The response generation in conversational agents also follows different techniques and can be retrieval-based or generative-based [16]. Most existing conversational agents are retrieval-based and generate responses using templates, where templates are predefined to hold variable information in the form of slots [12]. On the other hand, generative-based conversational agents are trained on vast amounts of end-to-end conversations and generate new responses using neural approaches [10].

Since both, rule-based and probabilistic/data-driven approaches have their drawbacks, hybrid approaches combine the two in one or more components of the dialog system to improve conversations in conversational agents [15, 17, 18].

2.2. Chatbots in Education

Chatbots in education have been popular already for a few years [19] since they offer a natural type of interaction to students to express themselves without fear of getting judged.

Application of conversational systems in e-learning encompasses diverse types of interaction such as being a counterpart in language learning [20], delivery of quizzes [21], triggering and steering student discussions in collaborative learning settings [22] or learning assistance by answering students' questions [23].

Conversational systems acting like human tutors in problem-based learning have been studied as well [24, 25]. In that context, it was found already very early that students learn better when a conversational system encourages them to explain answers (e.g., in mathematics [26]).

How to best represent domain knowledge has been studied extensively in the domain of intelligent tutoring systems (ITS). The type of representation often goes hand in hand with tutoring strategies, especially student solutions assessment.

For instance, so-called model-tracing approaches (e.g., as mentioned by Aleven [27]) model a correct solution to a given task and then measure a student's deviation from it.

For so-called ill-defined domains [28] (see also Section 3 below), acceptable solutions can be quite different from each other. Here, constraint-based modeling fits better, where all solutions are accepted that does not violate a set of predefined constraints.

Besides assessment, models of domain knowledge can also be used to generate questions, e.g., in conversational ITS. For instance, [29] use concept maps consisting of triples (which correspond to edges in a knowledge graph) and generate questions from them.

The question of how to use conversational ITS to trigger student reflection is less intensely studied. However, it has been shown that students' self-explanation improves learning outcomes, and that reflection and feedback could be essential building blocks in supporting students' self-regulated learning [30].

Our specific contribution in this context is the introduction of reflective dialog into group settings, specifically fostering individual reflection about collaborative work outcomes. We introduce a self-explanation mechanism that fits well for problem-based learning, e.g., in the domain of business information systems – we show how domain models that recursively decompose problems into sub-tasks can be used not only to discuss solutions "top-down", but also to encourage explanations for partial solutions.

3. Modeling domain knowledge: a hierarchical approach

As explained in the Introduction, we would like to build a conversational system that supports reflection about group assignments in the general domain of Business Information Systems (BIS).

In Intelligent Tutoring Systems (ITS), researchers have introduced the concept of "ill-defined domains" [31], characterized by the existence of multiple, sometimes even controversial solutions to a task which might all be considered "correct". While there can often be a wide range of "correct" solutions, acceptable solutions usually need to fulfil certain constraints – which is why ITS in ill-defined domains often rely on constraint checking for assessment [32].

The domain of BIS can be considered to be ill-defined in the above sense because of its typical tasks: in BIS, students frequently have to analyze situations ("cases"), relying on usually incomplete information – a typical property of an ill-defined task [28]. Based on the analysis of the situation and an understanding of the requirements involved in it, BIS students usually need to *design* a solution – a creative task that [28] also classify as ill-defined.

We have analyzed a range of tasks in the BIS domain and found the following subtasks to be frequent:

- *Requirements analysis:* given the description of a case, students need to identify and name the needs of business stakeholders
- *Selection of functionalities:* based on a set of available standard functionalities, students need to select those that will best satisfy the stakeholder needs.
- *(Technical) configuration:* while many BIS tasks do not require programming, there is usually a step where (technical) components realizing the functionalities as mentioned above need to be configured and combined.

While some of the above steps are shared with the process of software engineering, they can also be found in tasks that do not involve the design of software artifacts.

In fact, in some tasks, we found only some of these subtasks (e.g., some tasks skip the requirement analysis), or even different ones, but the principle of decomposition of tasks into subtasks was always present. We could identify it even when the teacher who had formulated the task did not make the principle explicit or was not fully aware of it. We will call it a "top-down" approach to solving BIS assignments and will show how to represent it with tree-like structures in the next section.

3.1. Graphical representation

A "top-down" solution to a student assignment in the BIS domain can be represented as a tree, where the root of the tree represents the entire assignment, and the branches at each level represent a way of decomposing the current subtask into further sub-subtasks.

We refer to the edges in this tree as "statements". Inspired by the triples used in semantic technologies such as RDF [33], a statement is defined as a triple (S, P, O), consisting of

- *A subject S* which represents the current (sub-)task T to be solved.

- *A predicate P* that stands for a typical activity done to solve *T*
- *An object O* which represents an element *E* of a solution for *T*. For instance, if *T* is a stakeholder requirement, *E* can be a functionality of an information system that helps to satisfy *T*.

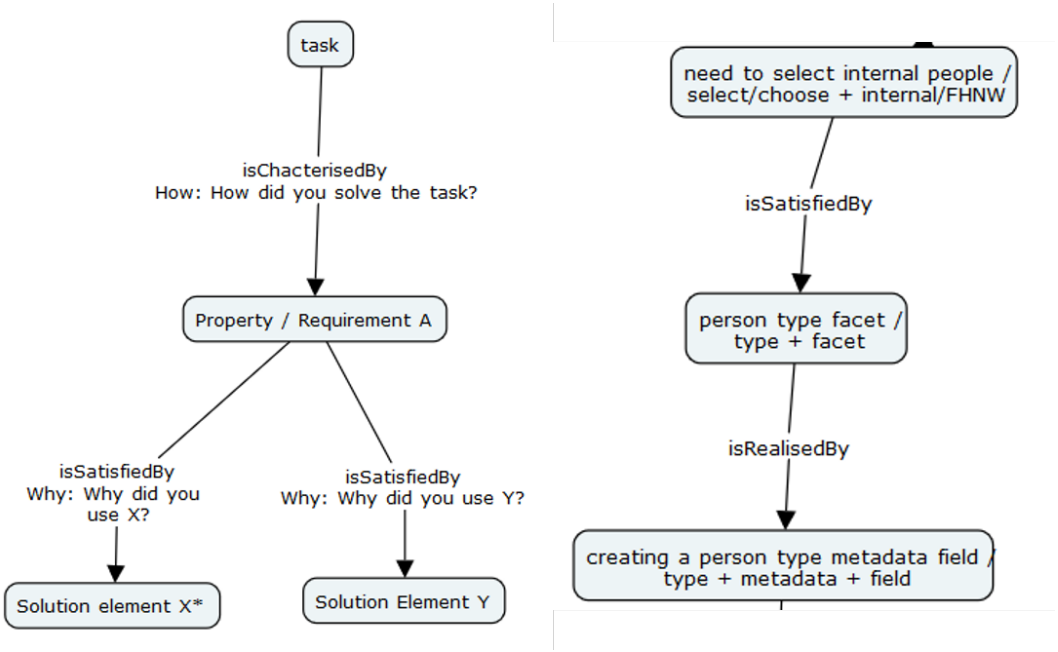

Figure 2: A general pattern for domain trees and an example tree model for the domain of enterprise search

This general pattern is illustrated on the left side of Figure 2, where the upper part contains a statement representing the identification of a requirement and the lower part shows how the requirement can be satisfied by two possible solution elements.

On the right side of Figure 2, an example snippet of a tree from the enterprise search domain is shown, including two statements. The assignment consists of configuring a search engine to support searching for persons. The top node in this subtree represents the need for a stakeholder to constrain the search to persons who are internal to the organization ("FHNW", in this case). The predicate and object of the upper statement define the solution of adding a "person type facet" to the search interface. The lower statement states that such a facet can be realized via a person type metadata field.

Several options can be used to encode further knowledge:

- Each node in the tree can have synonyms, separated by slashes ("/"), e.g. "select/choose".
- Below the name of a predicate the teacher can insert so-called prompts. These are questions or hints that should help students when they either fail to explain a solution by

mentioning the object ("How" prompts) or to justify a chosen solution by referring to the subject ("Why" prompts). See Section 4 for more details.

- Objects of a statement can be marked as mandatory with an asterisk ("*"), as it has been done for solution element X on the left side of Figure 2.

The last of these options can be used to produce entire branches of the tree, which are optional, i.e., that will only be discussed if the student mentions them. In general, students can mention, e.g., solution elements that go beyond the tree model, without being judged about it. This is intended, our aim being to trigger reflection about a chosen solution, not to judge the correctness of elements. However, students will always be asked for branches that are marked as mandatory, i.e., it is possible to enforce discussion about certain essential solution elements.

3.2. Ontology representation

We transform the graphical representation of the student assignments into an ontological representation of the domain concepts, the object properties, the "statements" and the prompts using a custom parser.

The knowledge base of the DSSA is developed as an ontology in RDFS [34] and in the Shapes Constraint Language (SHACL) [35]. The custom parser helps to adopt a model-driven approach to automate the generation of ontology elements, which, otherwise, would require a considerable knowledge engineering effort.

The *subjects* and *objects* classes are declared in the didactics domain, while their individuals are declared in dedicated tutoring domains as elements, e.g., digitag:BeautifulPicture, digibp:ServiceTask, where "digitag" and "digibp" represent the tutoring domains of the student assignments. *Predicates* (e.g., digitag:isDrawnBy) are declared as object properties having the *subjects* and *objects* as domains and ranges. An example of a "statement" is as shown below:

```
digitag:Statement_1
    rdf:type didactics:Statement ;
    didactics:subject digitag:BeautifulPicture ;
    didactics:predicate digitag:isDrawnBy ;
    didactics:object digitag:Flower ;
    didactics:object digitag:Meadow ;
    dssa:StatementHasHowPrompt digitag:Prompt_1 ;
.
```

A *SHACL NodeShape* is used to constrain the definition of a *statement* to have exactly one *subject*, at least one *object* and exactly one *predicate*. SHACL rules are also declared in the NodeShape to automatically infer the *status* as "active-subject" or "active-object" (see Section 4 for further explanation). These SHACL rules infer new knowledge using SPARQL queries defined with sh:SPARQLRule, namely, which "statements" have been entirely or partially included in the students' utterances, the expected domain concepts and the prompts in the case of partial statements. The inferred knowledge is utilized in the dialog management to decide the next action and response.

4. Dialog management

How should students reflect on typical BIS tasks? As we have seen in Section 3.1 above, a top-down approach to solving BIS assignments leads to a tree-like structure (see Figure 2). We believe that, starting at any position in such a tree, students should be able to reflect on their knowledge in both directions, namely

1. "downwards", by explaining how to solve the current subtask by decomposing it into further subtasks – e.g., by naming a functionality that will help to satisfy a particular requirement – and
2. "upwards", by justifying why the current subtask is necessary and why it solves its predecessor in the tree – e.g., by justifying a chosen functionality in terms of the requirement(s) that it satisfies.

To trigger these types of reflection, we employ the following dialog management strategies:

- **Intent recognition:** In our system, intents to be recognized are equal to "statements" and the nodes in the domain tree represent the slots to be filled in from the student utterances. In our first prototype of DSSA, we employ a simple entity recognition strategy by creating a custom Named Entity Recognition (NER) tagger, using the CRFClassifier from the Stanford NLP library [36]. Sample dialogs are used to train the custom NER tagger by labeling the keywords in the dialogs with the URI of the corresponding nodes in the domain tree. Whenever the keywords used as labels of the domain tree nodes are found in a student's utterance, the utterance is annotated with the URI of the corresponding node. In the future, we plan to extend the annotation and recognition of domain-specific keywords by applying more sophisticated methods, see Section 5.1.

- **Reasoning:** Using ontology inferencing based on SHACL rules (see Section 3.2), the system determines all triples (S, P, O) where either the subject S has been annotated in the student's utterance or the object O or both. These triples are henceforth called *active*. Triples, where only the subject S was mentioned, are called "active-subject", ones where only O was mentioned "active-object".

- **Generation of questions**
  - **Downward step:** For "active-subject" statements, the system generates the following question: "How did you <P><S>?", where <P> is the active voice of the predicate P. For instance, if a student mentions the keywords "select" and "internal", a system using the right-hand side of Figure 2 as a domain tree will generate the question "How did you satisfy the need to select internal people?"
  - **Upward step:** For "active-object" statements, the system generates the question "Why did you use <O>?". For the example of mentioning the person type facet in Figure 2, that question will read "Why did you use a person type facet?"

The downward and upward steps correspond to the above-mentioned abilities of students to find solutions (downward) and justify them (upward). A final part of the dialog management strategy consists of deciding which question should be asked next. After each student utterance, the following steps are applied:

1. All newly activated statements S, P, O will be put on a stack. Thus, the next statement to be covered will be the one that has last been put on the stack (last-in-first-out, LIFO). "Active-subject" statements are pushed first to the stack, followed by "active-object" statements. Because of the LIFO principle of the stack, the "active-object" statements are hence retrieved first in step 3.
2. Statements S, P, O are deleted (while on the stack or before being pushed there) if both S and O have been mentioned by the student at any point of the conversation. They are then marked as "covered".
3. Then, the statement S, P, O that is currently on top of the stack is retrieved, and questions are generated. For an "active-subject" statement, which is marked as mandatory, a downward question will be generated automatically as described above. The system keeps a memory of all questions asked so far – if the question has been asked already, it will try to retrieve a "how" prompt. If the statement on top of the stack is "active-object" a downward question or "why" prompt will be generated analogously – no matter if the object of the statement is mandatory or not. If all questions and prompts have been asked already, the statement will be marked as "given up" and removed from the stack. In this case, this third step is repeated, i.e., the next statement is fetched from the top of the stack.
4. If there are no more statements on the stack, the system ends the conversation.

The reason for adopting these choices is as follows: When a tutor discusses an assignment solution with a student, the conversation usually starts with some part of a solution. The tutor then asks for more details (downward questions). When such an aspect of a solution consists of several parts – i.e., several branches of a domain tree – it is natural to first finish discussing about one branch before addressing the next one. This can be realized by using a stack since it results in a depth-first search of the domain tree (when considering only downward steps). A breadth-first search would result in jumping between branches all the time.

When a student enters the discussion with some rather detailed part of a solution – i.e., jumps ahead to a lower part of the domain tree, leaving out some intermediate aspects of a solution – it is natural to ask an upward question, i.e., to let the student justify the choice in terms of the intermediate aspects.

4.1. Illustration

In this section, we illustrate the principles described above with an example dialog. We have chosen a simple task for easy understanding – namely, painting a picture with a green lawn and a purple flower on it, using only the primary watercolors blue, red, and yellow. The task's decomposition is shown on the left side of Figure 3, an example dialog on its right side.

The flow of the conversation shows how the system guides the student to decompose the task. It first covers the tree's left branch (drawing the flower), asking how the flower is colored and how to mix purple.

Since all objects/branches of the tree are mandatory, it then asks "Can you say which other objects you had to draw" – which is a how prompt attached to the root node. That is, the student should also cover the right branch of the tree. In this case, however, the student jumps ahead to a leaf node ("yellow"), which the system answers with a "why" question to get a justification for the use of yellow.

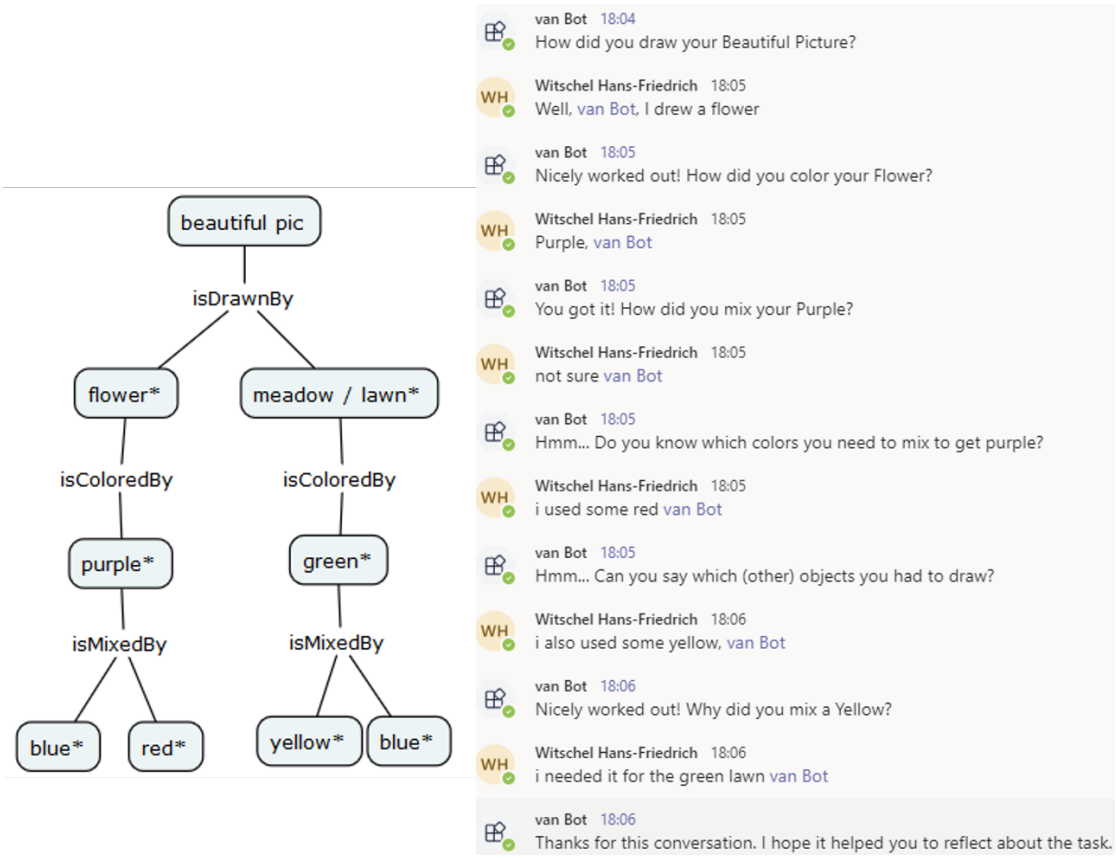

Figure 3: A toy example of a domain tree and a corresponding conversation

4.2. DSSA Prototype

The prototype of DSSA is decomposed into several components described in previous sections:

- a specification of a modeling language that will help the lecturers to graphically model the cognitive requirements of their assignments
- a custom parser that translates the tree-like model into DSSA's knowledge structure
- a dialog manager
- an instance of a chatbot deployed in a Microsoft Teams channel

The components of DSSA were evaluated with different stakeholders including lecturers, students and the members of digital learning team. The feedback, received at different stages, has been partly incorporated in the prototype and partly planned as future improvements described in the next section.

5. Learning smoother conversations

Our current approach works quite well with simple assignments. The custom NER tagger annotates simple domain-specific keywords and phrases. However, the capability to identify the intent from the underlying meaning of the utterance is currently limited. The response generation is retrieval-based using templates. As a result, one can notice that some responses like "Why did you mix a Yellow?" (see Figure 3) seem awkward. To make the conversations smoother, we plan to adopt hybrid approaches that will enhance the understanding and the response generation of DSSA.

5.1. Natural language understanding

In the example in Figure 3, domain concepts can be identified and annotated easily. Although for more complex domains, there could be several different ways of answering the same question. In these situations, annotating the domain concepts is not straightforward. For instance, given a question "Which stakeholders did you identify for this task?" the student might respond – "We chose internal people at FHNW" or "We chose people who are working internally at FHNW" or "We chose internal employees of FHNW" (see Figure 2). For each of these utterances, DSSA should be able to identify the same domain concept. The understanding of DSSA in similar situations can be improved once we have sufficient training data to identify complex domain concepts from student utterances.

Our custom annotations can be further enriched by enhancing the NLU pipeline through the addition of components like Part of Speech (POS) tagger that can identify fine-grained verbs translating to the predicates in our domain ontology. An additional approach would be to identify not just the domain concepts, but an entire statement by annotating the subject, predicate, and object as defined in the domain ontology. Statement annotations would go a long way in relieving the load of our reasoner, which then would be needed only to infer partial statements from the student utterances.

5.2. Natural language generation

A popular technique to generate responses in conversational agents is to use sequence to sequence models. Sequence to sequence models are based on neural networks that are well-known for their requirement of very high training data. However, domain-specific situations usually have very limited training data and a few approaches have tackled this issue [37, 38]. The training data produced by interacting with DSSA will be very specific to the assignment in a particular topic and hence quite limited in its overall training capability. Leveraging the domain-specific concepts in our ontology in combination with a sequence to sequence model similar to [39] can prove beneficial to train DSSA.

6. Conclusions and future work

In this paper, we discussed a conversational approach to trigger individual student reflection regarding group assignments in BIS. The teachers start by modeling the assignment in a tree-like

representation using a top-down approach. The knowledge about the assignments is transferred from the model into an ontology to represent the key domain concepts and the relationships between the concepts. We annotate the student utterances in sample dialogs with the URIs of the domain concepts. The conversational agent DSSA then uses ontology inferencing to verify the student's understanding and determine the details that were not covered by the student. The dialog manager component of DSSA applies this information to determine the next action in the dialog policy and carries out an end-to-end conversation with the student.

Besides enhancing the conversational ability of DSSA, we also plan to introduce sentiment and emotion recognition from student utterances. It has been observed that student emotions, motivation, and cognition are linked to each other [40, 41, 42]. Taking student emotions into consideration, DSSA can adapt its conversation strategy to focus on specific areas that would benefit the students by reflecting on their motivation and learning.

References

[1] G. Brown, J. Bull, M. Pendlebury, Assessing Student Learning in Higher Education, Psychology Press, 1997.

[2] N. Webb, Assessing students in small collaborative groups, Theory into practice 36 (1997) 205 – 213.

[3] L. M. Harding, Students of a feather "flocked" together: A group assignment method for reducing free-riding and improving group and individual learning outcomes, Journal of Marketing Education 40 (2018) 117–127.

[4] S. C. Lambert, A. J. Carter, M. Lightbody, Taking the guesswork out of assessing individual contributions to group work assignments, Issues in Accounting Education 29 (2014) 169–180.

[5] M. C. English, A. Kitsantas, Supporting student self-regulated learning in problem-and project-based learning, Interdisciplinary journal of problem-based learning 7 (2013) 6.

[6] A. D. Gertzman, J. L. Kolodner, A case study of problem-based learning in a middle school science classroom: Lessons learned, in: Proceedings of the 1996 international conference on Learning sciences, International Society of the Learning Sciences, 1996, pp. 91–98.

[7] J. Archer-Kath, D. W. Johnson, R. T. Johnson, Individual versus group feedback in cooperative groups, The journal of social psychology 134 (1994) 681–694.

[8] J. Gao, M. Galley, L. Li, Neural approaches to conversational ai, in: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, 2018, pp. 1371–1374.

[9] J.-G. Harms, P. Kucherbaev, A. Bozzon, G.-J. Houben, Approaches for dialog management in conversational agents, IEEE Internet Computing 23 (2018) 13–22.

[10] S. Hussain, O. A. Sianaki, N. Ababneh, A survey on conversational agents/chatbots classification and design techniques, in: Workshops of the International Conference on Advanced Information Networking and Applications, Springer, 2019, pp. 946–956.

[11] P. B. Brandtzaeg, A. Følstad, Why people use chatbots, in: International Conference on Internet Science, Springer, 2017, pp. 377–392.

[12] D. Jurafsky, J. H. Martin, Dialogue systems and chatbots, Speech and Language Processing (2019).
[13] M. McTear, Z. Callejas, D. Griol, Introducing the conversational interface, in: The Conversational Interface, Springer, 2016, pp. 1–7.
[14] Google.com, Dialogflow, 2020. URL: https://cloud.google.com/dialogflow, (Accessed on 01/11/2020).
[15] T. Bocklisch, J. Faulkner, N. Pawlowski, A. Nichol, Rasa: Open source language understanding and dialogue management, arXiv preprint arXiv:1712.05181 (2017).
[16] K. Ramesh, S. Ravishankaran, A. Joshi, K. Chandrasekaran, A survey of design techniques for conversational agents, in: International Conference on Information, Communication and Computing Technology, Springer, 2017, pp. 336–350.
[17] J. D. Williams, K. Asadi, G. Zweig, Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning, arXiv preprint arXiv:1702.03274 (2017).
[18] X. Li, Y.-N. Chen, L. Li, J. Gao, A. Celikyilmaz, End-to-end task-completion neural dialogue systems, arXiv preprint arXiv:1703.01008 (2017).
[19] A. Kerry, R. Ellis, S. Bull, Conversational agents in E-Learning, in: International Conference on Innovative Techniques and Applications of Artificial Intelligence, 2008, pp. 169–182.
[20] L. K. Fryer, M. Ainley, A. Thompson, A. Gibson, Z. Sherlock, Stimulating and sustaining interest in a language course: An experimental comparison of chatbot and human task partners, Computers in Human Behavior 75 (2017) 461–468.
[21] J. Pereira, Leveraging chatbots to improve self-guided learning through conversational quizzes, in: Proceedings of the fourth international conference on technological ecosystems for enhancing multiculturality, 2016, pp. 911–918.
[22] S. Tegos, S. Demetriadis, Conversational agents improve peer learning through building on prior knowledge, Journal of Educational Technology & Society 20 (2017) 99–111.
[23] M. Coronado, C. A. Iglesias, Á. Carrera, A. Mardomingo, A cognitive assistant for learning java featuring social dialogue, International Journal of Human-Computer Studies 117 (2018) 55–67.
[24] A. M. Olney, A. C. Graesser, N. K. Person, Tutorial dialog in natural language, in: R. Nkambou, R. Mizoguchi, J. Bourdeau (Eds.), Advances in intelligent tutoring systems, Springer Science and Business Media, Berlin/Heidelberg, 2010, pp. 181–206.
[25] A. M. Olney, S. D'Mello, N. Person, W. Cade, P. Hays, C. Williams, B. Lehman, A. Graesser, Guru: A computer tutor that models expert human tutors, in: International conference on intelligent tutoring systems, Springer, 2012, pp. 256–261.
[26] V. Aleven, K. R. Koedinger, K. Cross, Tutoring Answer Explanation Fosters Learning with Understanding, in: Artificial Intelligence in Education, IOS Press, 1999, pp. 199–206.
[27] V. Aleven, Rule-based cognitive modeling for intelligent tutoring systems, in: R. Nkambou, R. Mizoguchi, J. Bourdeau (Eds.), Advances in intelligent tutoring systems, Springer Science and Business Media, Berlin/Heidelberg, 2010, pp. 33–62.
[28] C. F. Lynch, K. D. Ashley, V. Aleven, N. Pinkwart, Defining ill-defined domains; a literature survey, in: Intelligent Tutoring Systems (ITS 2006): Workshop on Intelligent Tutoring Systems for Ill-Defined Domains, 2006.

[29] A. M. Olney, A. C. Graesser, N. K. Person, Question generation from concept maps, Dialogue & Discourse 3 (2012) 75–99.
[30] G. Van den Boom, F. Paas, J. J. Van Merrienboer, T. Van Gog, Reflection prompts and tutor feedback in a web-based learning environment: Effects on students' self-regulated learning competence, Computers in Human Behavior 20 (2004) 551–567.
[31] P. Fournier-Viger, R. Nkambou, E. M. Nguifo, Building intelligent tutoring systems for ill-defined domains, in: R. Nkambou, R. Mizoguchi, J. Bourdeau (Eds.), Advances in intelligent tutoring systems, Springer Science and Business Media, Berlin/Heidelberg, 2010, pp. 81–101.
[32] A. Mitrovic, Modeling domains and students with constraint-based modeling, in: R. Nkambou, R. Mizoguchi, J. Bourdeau (Eds.), Advances in intelligent tutoring systems, Springer Science and Business Media, Berlin/Heidelberg, 2010, pp. 63–80.
[33] F. Manola, E. Miller, B. McBride, et al., Rdf primer, W3C recommendation 10 (2004) 6.
[34] D. Allemang, J. Hendler, Semantic web for the working ontologist: effective modeling in RDFS and OWL, Elsevier, 2011.
[35] H. Knublauch, D. Kontokostas, Shapes constraint language (shacl), W3C Candidate Recommendation 11 (2017).
[36] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, D. McClosky, The stanford corenlp natural language processing toolkit, in: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, 2014, pp. 55–60.
[37] J. Kapočiūtė-Dzikienė, A domain-specific generative chatbot trained from little data, Applied Sciences 10 (2020) 2221.
[38] J. Kim, H.-G. Lee, H. Kim, Y. Lee, Y.-G. Kim, Two-step training and mixed encoding-decoding for implementing a generative chatbot with a small dialogue corpus, in: Proceedings of the Workshop on Intelligent Interactive Systems and Language Generation (2IS&NLG), 2018, pp. 31–35.
[39] R. L. Logan IV, N. F. Liu, M. E. Peters, M. Gardner, S. Singh, Barack's wife hillary: Using knowledge-graphs for fact-aware language modeling, arXiv preprint arXiv:1906.07241 (2019).
[40] E. A. Linnenbrink, Emotion research in education: Theoretical and methodological perspectives on the integration of affect, motivation, and cognition, Educational psychology review 18 (2006) 307–314.
[41] D. K. Meyer, J. C. Turner, Re-conceptualizing emotion and motivation to learn in classroom contexts, Educational Psychology Review 18 (2006) 377–390.
[42] P. Op't Eynde, J. E. Turner, Focusing on the complexity of emotion issues in academic learning: A dynamical component systems approach, Educational Psychology Review 18 (2006) 361–376.