Basic tutorial tactics for virtual agents. DynaLearn, EC FP7 STREP project 231526, Deliverable D5.2

Wißner, M.; Häring, M.; Mehlmann, G.; Bühling, R.; Milosevic, U.; Liem, J.; Linnebank, F.; Beek, W.; Bredeweg, B.; André, E.

Publication date
2010

Document Version
Final published version

Link to publication

Citation for published version (APA):
Wißner, M., Häring, M., Mehlmann, G., Bühling, R., Milosevic, U., Liem, J., Linnebank, F., Beek, W., Bredeweg, B., & André, E. (2010). *Basic tutorial tactics for virtual agents. DynaLearn, EC FP7 STREP project 231526, Deliverable D5.2*. DynaLearn. http://hcs.science.uva.nl/projects/DynaLearn/DeliverablesPublic/D5.2.pdf

Deliverable number:	D5.2
Deliverable title:	Basic Tutorial Tactics for Virtual Agents

Delivery date:	2010/07/31
Submission date:	2010/08/31
Leading beneficiary:	Augsburg University (UAU)
Status:	Version 06 (final version)
Dissemination level:	PU (public)
Authors:	Michael Wißner, Markus Häring, Gregor Mehlmann, René Bühling, Urosh Milosevic, Jochem Liem, Floris Linnebank, Wouter Beek, Bert Bredeweg, Elisabeth André

Project number:	231526
Project acronym:	DynaLearn
Project title:	DynaLearn - Engaging and informed tools for learning conceptual system knowledge
Starting date:	February 1st, 2009
Duration:	36 Months
Call identifier:	FP7-ICT-2007-3
Funding scheme:	Collaborative project (STREP)

Abstract

This document presents the progress and effort made in the design of a dialog system for the virtual characters in DynaLearn. The main purpose of this system is to provide means by which the virtual characters can present relevant system knowledge to the learners in a pedagogically sound manner.

After providing an overview of the role of dialogs in interactive learning environments we present the architecture of our approach and describe in detail the functionality of its three main components: Dialog Management, Verbalization and User Modeling.

Internal reviewers

Jorge Gracia del Río (UPM), Ontology Engineering Group, Universidad Politécnica de Madrid

Andreas Zitek (BOKU), Institute of Hydrobiology and Aquatic Ecosystem Management, University of Natural Resources and Applied Life Sciences

Acknowledgements

The Bayesian models depicted in this paper were created using the GeNIe modeling environment and the core of our User Model implementation is based on jSMILE, a Java wrapper for the SMILE reasoning engine for graphical probabilistic models. Both GeNie and SMILE are developed and contributed to the community by the Decision Systems Laboratory, University of Pittsburgh (http://dsl.sis.pitt.edu).

For the parsing of OWL-based data, we use the OWL API, primarily maintained by the University of Manchester (http://owlapi.sourceforge.net).

Document History

Version	Modification(s)	Date	Author(s)
01	First version	2010-07-14	Wißner,Häring,Mehlmann,Bühling
02	Input from UvA	2010-07-16	Milosevic,Liem,Linnebank,Beek,Bredeweg
03	Draft for review by partners	2010-07-19	Wißner,André
04	Integrated suggestions from review by UPM	2010-07-26	Wißner,Bredeweg
05	Integrated suggestions from review by BOKU	2010-08-16	Wißner
06	Final version	2010-08-23	Wißner,Bühling

Contents

1. Introduction

The DynaLearn project aims to conceive and develop an interactive learning environment which combines current technologies and research from different areas in a way that provides learners and teachers with the optimal tools for a rich educational experience.

The DynaLearn software reflects this through the integration of the following three modules, each providing different benefits to the overall application:

- **Conceptual Modeling (CM): Offers a graphical editor to build diagrammatic representations to learners to articulate, analyze and communicate ideas, and thereby construct their conceptual knowledge**
- **Semantic Technology (ST): Provides web-based ontology mapping which be used to find and match co-learners working on similar ideas to provide individualized and mutually benefiting learning opportunities**
- **Virtual Characters (VC): A cast of different virtual characters can be called upon to make the interaction with the software engaging and motivating**

To further specify the scope of this report, work on the Virtual Characters consists of the following tasks:

- **To enable learners to express their ideas on a conceptual model using a virtual character as a presenter that combines verbal and non-verbal means for communication**
- **To realize various kinds of dialog between virtual characters representing different roles and functions to explain a conceptual model**
- **To design communicative strategies for multiple agents that engage in a dialog about the model created by their learners**

This Deliverable reports on the progress of work on the last item, namely the results of Task 5.2 “Design and implementation of dialog functionality for characters”.

The remainder of this document is structured as follows: We will first give an overview of the state of art in the field of dialog management for learning environments (Chapter 2). We will then show the current architectural state of the parts of the software relevant for this deliverable (Chapter 3). After that, we will elaborate on the different components of the architecture (Chapters 4-6) and we will end with a conclusion (Chapter 7) as well as a discussion and an outlook of future work (Chapter 8).

2. State of the Art/Literature

2.1. Dialog Modes in Tutoring

For a believable behavior of a virtual character with a positive impact on learning success we need to know how human teachers accomplish this task. Cade et al. observed and analyzed professional tutors at work. They identified eight mutually exclusive dialog modes (extracted from Cade et al. [1]):

- Introduction: This mode captures the non-teaching dialogue that occurs as both tutor and student exchange greetings and attempt to establish the agenda for the tutoring session.
- Lecture: The tutor explicitly dispenses domain information to the student.
- Highlighting: During problem-solving activities, the tutor highlights what the problem is asking for and the information that it provides. The tutor can also break down the steps involved in a particular problem-solving method, create a "game plan" for working a problem, and redirect the student to the right path when they have forgotten parts of the method and its purpose.
- Modeling: In this mode, the tutor works out a problem for the student. While the student may "participate" in the problem-solving process (i.e., do minor calculations that the tutor instructs them to do), the tutor takes the lead in solving the problem and initiates all of the problem-solving steps.
- Scaffolding: The tutor and student work out a selected problem together, each contributing portions of the answer that result in a solution. The tutor intervenes when necessary so that the student can successfully arrive at a correct solution.
- Fading: The student works an entire problem with virtually no aid from the tutor, although the tutor may comment from time to time on their progress.
- Off Topic: This is a portion of the session where the tutor and student are not engaged in the tutoring lesson for a significant amount of time.
- Conclusion: Like the *Introduction*, this mode captures the social pragmatics as the session comes to a close, which has little to do with the lesson.

The study also included analyses about the average frequency and the likeliness of transition between the modes. On average each tutoring session consisted to over 50% of lecture and scaffolding. Also a lot of transitions between those modes were observed. So lecture and scaffolding were identified as the most important elements. Later the expert tutors confirmed that they prefer "learning by doing" over modeling. Highlighting was mostly used in combination with scaffolding, to get the student back on track.

Those results back up our assumption, that interaction is an important element in successful learning. That is why we concentrated our further research on scaffolding and how to integrate this into an Intelligent Learning Environment.

2.2. Scaffolding

Lindsay Lipscomb and colleagues summarized some scientific opinions on scaffolding, which describe it as a more extensive teaching principle [2] than Cade et al. who see scaffolding as one of their mutual exclusive dialog modes.

Lipscomb et al. first describe Lev Vygotsky's *Zone of Proximal Development* [3]. Vygotsky divides a learner's developmental level into two parts: the *actual developmental level,* which represent the capability of the learner to solve problems without help and the *potential developmental level*, which the learner can reach under guidance of a more competent helper. The *Zone of Proximal Development (ZPD*) is the zone between those two levels. A good teacher challenges his students with tasks that are in the ZPD. Otherwise they might be unchallenged or not able to solve the problem at all. While learning in the ZPD the *actual developmental level* rises and so does the *potential developmental level.* This means of course that also the *Zone of Proximal Development* changes all the time and the teacher has to adjust his means.

Vygotsky also emphasizes the importance of an independent working learner. The task of the teacher is to assess the skills of the learner to confront him with problems, he can solve with only as much help as necessary. With growing knowledge the learner should need less and less guidance. Vygotsky calls this process *fading*, in contrast to the definition of Cade et al. where fading is one of their dialog modes. You can say the process of Vygotsky ends in the dialog mode of Cade et al.

Though Vygotsky never used the term *Scaffolding,* his work is often cited in this context. Because his work describes the fundamentals of this teaching principle: learner-adjusted guidance, minimality of the means, actively participation of the learner during the learning process and the resulting alternating activity of teacher and learner.

For a more sophisticated description Lipscomb et al. cite the six general elements of scaffolded instructions of Zhao et al. [4].

- Sharing a Specific Goal: Teacher and learner have to share the same goal. It lies in the responsibility of the teacher to establish this goal, considering the needs and interests of the learner. For this he has to assess the student's current knowledge and skills.
- Whole Task Approach: the learning process is focused on the overall goal. The student has to be taught everything he needs to solve the actual task.
- Immediate Availability of Help: it is important to keep motivation and self-efficacy high. The teacher has to ensure that the learner achieves consistent successes by offering help at the right moment and when it is really needed.
- Intention-Assisting: the teacher has to ensure that the learner follows effective strategies. Is this not the case he must act and bring the learner back on track. If the learner does everything right, this status should not be disturbed and help should only be provided following the third element of scaffolded instructions.
- Optimal Level of Help: one of the greatest challenges for the teacher is to find the balance between guidance and independent work of the learner. The teacher should only help if the learner is not able to cope with the current problem, to prevent frustration. But he should not help too much, so the learner still plays an important part in the learning process.
- Conveying an Expert Model: the teacher should provide expert models for orientation. The learner has to understand how this expert ways are accomplished.

With this Zhao et al. define the basics for successful scaffolding. Lipscomb et al. additionally provide a set of methods a teacher can use to convey the necessary information: modeling the desired behavior, providing explanations, ask the learner to participate, evaluating the understanding of the learner and asking the learner to provide own solutions. This already points out that Lipscomb et al. see dialog modes like modeling and highlighting, according to Cade et al., as part of scaffolding and hence broaden the definition of this term. According to the references of Lipscomb et al. the teaching process should be started with modeling. The teacher should show the learner how to solve such problems correctly,

verbalizing his thoughts. In the further process the learner should be involved more and more, enabling the teacher to evaluate the learner's knowledge and provide feedback.

With this scaffolding gets a relatively clear structure. But Cade et al. disagree with the theories, which see dependence in the order of dialog modes. Their results indicate that there is no dependence. Also Graesser et al. [5] noticed in their observations of tutors, that they do not follow the scientific ideal strategies and still are successful.

It is obvious that there is consensus in scaffolding in regard to the basics described by Vygotsky or at least similar approaches but there is no optimal way. But this provides us with the freedom, to find a way that considers the facts depicted by Lipscomb et al. and uses the dialog modes of Cade et al.

2.3. Step-based feedback

The next step was to get an idea how to implement scaffolding for an intelligent tutoring system. One of the best known intelligent tutoring systems is AutoTutor from Graesser et al. Its main goal is to teach students a deeper understanding of the subject matter and help them to give more detailed answers. For this a virtual character asks the student a series of questions and tries to establish an ongoing dialog.

AutoTutor follows a 5-step dialog system, a pattern Graesser et al. retrieved from their observations of human expert tutors.

Step 1: Tutor asks question (or presents problem)
Step 2: Learner answers question (or begins to solve problem)
Step 3: Tutor gives short immediate feedback on the quality of the answer (or solution)
Step 4: The tutor and learner collaboratively improve the quality of the answer.
Step 5: The tutor assesses the learner's understanding of the answer

This structure of this step-based approach is very simple and hence good to implement for a computer system. It fits, especially with step 4, the scaffolding approach. With step 1 and 2 the tutor assesses the current state of the learner's knowledge. Step 3 establishes the shared goal namely to improve the answer of the learner. The last step evaluates the success of the process.

The dialogs are created using fuzzy production rules and the so called Dialogue Advancer Network (DAN). Figure 1 shows the DAN used in AutoTutor.

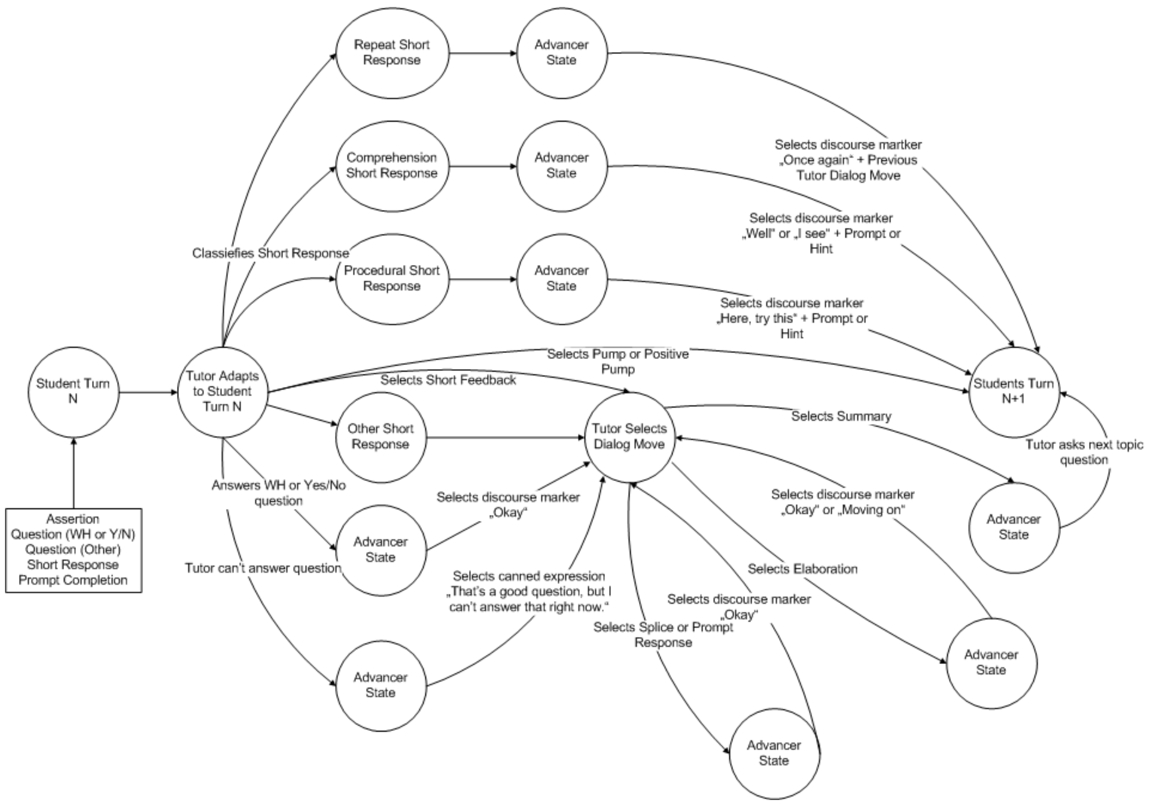

Repeat Short Response
Advancer State
Comprehension Short Response
Advancer State
Procedural Short Response
Advancer State
Classiefies Short Response
Selects discourse martker „Once again" + Previous Tutor Dialog Move
Selects discourse marker „Well" or „I see" + Prompt or Hint
Selects discourse marker „Here, try this" + Prompt or Hint
Selects Pump or Positive Pump
Student Turn N
Tutor Adapts to Student Turn N
Selects Short Feedback
Students Turn N+1
Other Short Response
Tutor Selects Dialog Move
Selects Summary
Tutor asks next topic question
Assertion Question (WH or Y/N) Question (Other) Short Response Prompt Completion
Answers WH or Yes/No question
Selects discourse marker „Okay"
Advancer State
Selects discourse marker „Okay" or „Moving on"
Advancer State
Tutor can't answer question
Selects canned expression „That's a good question, but I can't answer that right now."
Selects Elaboration
Selects discourse marker „Okay"
Selects Splice or Prompt Response
Advancer State
Advancer State
Advancer State

Figure 1: The Dialogue Advancer Network used in AutoTutor (Source: [5])

Using these DANs, AutoTutor combines different dialog steps to one ongoing dialog. The dialog steps belong to one of the following categories: main questions, short feedback, pump, prompt, prompt response, hint, assertion, correction or summary.

The fuzzy production rules determine with the help of different parameters which dialog step is chosen. Graesser et al. introduced the DAN because the rules do not consider the dialog history which resulted in confusing dialogs. Sometimes the students were not sure which action the agent was expecting from them.

The Dialogue Advancer Network consists of nodes and edges, very similar to a state machine. With the DAN, AutoTutor considers preceded dialog steps resulting in sound dialogs that convey the learners what they have to do.

Although AutoTutor has other goals than the tutoring elements of DynaLearn the technical ideas can also be applied to our needs. The use of state machines (or a similar structure like the DAN) to create a context sensitive dialog management also allows reacting fast on interaction of the learner with the system. As scaffolding relies on interaction between learner and tutor this is an important feature an appropriate intelligent tutoring system.

AutoTutor also shows the complexity of such state machines is manageable. With a suitable development environment teachers, even with low programming knowledge, should be able to adjust the behavior of the tutoring agents according to their needs.

That is why we decided to base our dialog management on *Hierarchical Concurrent State Machines* and the IDE SceneMaker. This kind of state machines allows us to create intuitive representations of agent behaviors implementing the scaffolding principle in regard to the research of Lipscomb et al. and Cade et al.

3. Architectural Overview

Before we start to describe the different components involved in the dialog functionality for the virtual characters, we will first, as reminder, give an overview of the architecture as described in Deliverable 2.1 "Technical design and architecture". The relevant parts of this architecture can be seen in Figure 2.

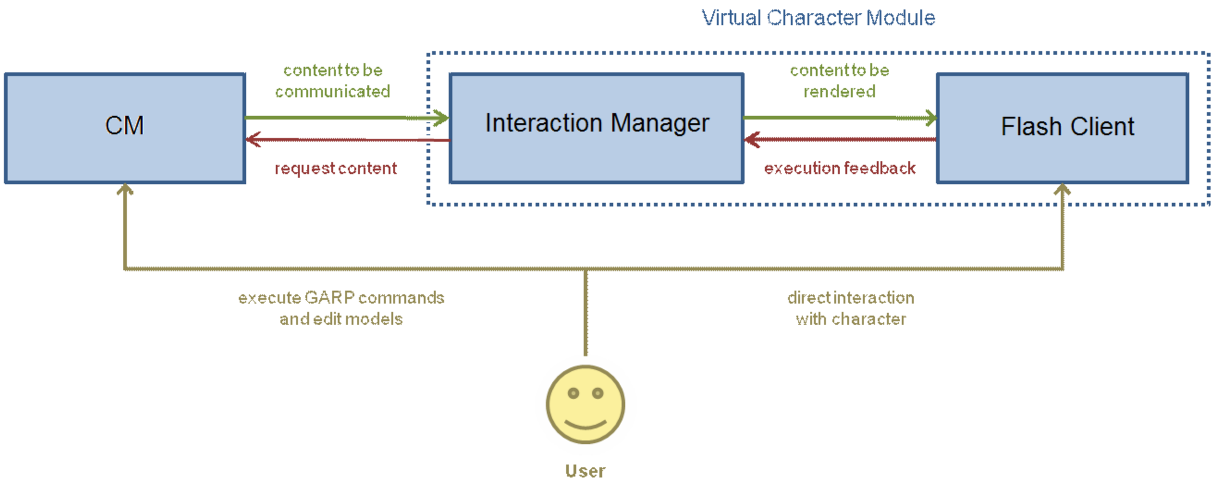


Figure 2: Architectural Overview of dataflow between CM and VC components

As we can see, the VC component consists of two modules, of which only one, the Interaction Manager (IM), directly communicates with the CM component. It acts as a middleware between the conceptual modeling part of the DynaLearn software and the Flash Client which actually displays the virtual characters and their actions. The IM mediates between the others by relaying messages, requests and feedback, with the two most common being interactions by a learner and the passing of conceptual knowledge.

The IM's main purpose however is the dialog functionality, i.e. the transformation of this conceptual knowledge in its formal representation into a screenplay that virtual characters can follow to present the knowledge to a learner in an understandable and engaging way. These screenplays are XML-based and prompt the characters to walk around, play animations and of course speak. The characters' utterances are displayed with speech bubbles, but can also be synthesized with the Mary Text-To-Speech System (http://mary.dfki.de/).

The IM's dialog functionality builds on the following three different parts, each of which will be further explained in the remainder of this document:

- **A Dialog Manager that decides *What to say*, based on learner input, dialog history, available data and other factors. Also, in our case of different characters, it must also be determined which character(s) should act.**
- **A Verbalizer that decides *How to say it*, i.e. what words to use.**
- **A User Model that keeps track of the user's knowledge and interactions. The data provided by it can act as a filter or decision criteria for both of the other parts.**

4. Dialog Management

Dialogs displayed by the virtual characters in the DynaLearn environment have to meet several requirements:

- **They should be reusable**
- **They should be parametrizable**
- **They might require input from other components at runtime**
- **They should support multiple languages**
- **They should be editable by non-computer experts**

Following these requirements, we decided to use SceneMaker, a toolkit for the authoring of interactive performances by virtual characters for the dialog management part in our architecture.

In this section, we will first describe the SceneMaker and its functionality and then explain how we used it in the context of the DynaLearn software.

4.1. The SceneMaker

4.1.1. Introduction

For the dialog management and interaction design of virtual characters' behavior, there are often employed *authoring tools*, whose main task is to allow inexperienced authors with low expert knowledge and little programming skills to rapidly prototype a model description for the virtual character's behavior.

Since rule-based, frame-based, plan-based, or agent-based approaches to dialog management and interaction design require expert knowledge in a multitude of domains [6], many authoring tools pursue a *finite-state-based* approach [12, 11, 9, 7], using some kind of *state-transition diagram*, as an intuitive and suitable method to describe the behavior of a virtual character on a high level of abstraction.

However, most of these state-of-the-art authoring tools use a single state-transition diagram, in which the behavior of many virtual characters, all their behavioral aspects, functions and modalities as well as the interactions between them and the user and among each other, respectively, are highly interwoven. As a result, it is a complex task for an author to identify, extract and understand the individual behavioral aspects and their coherences, making it error prone to modify single aspects in the model. Adding additional behavioral aspects or further virtual characters to the model may lead to an exponential explosion of the state space and to highly complex transition conditions [8].

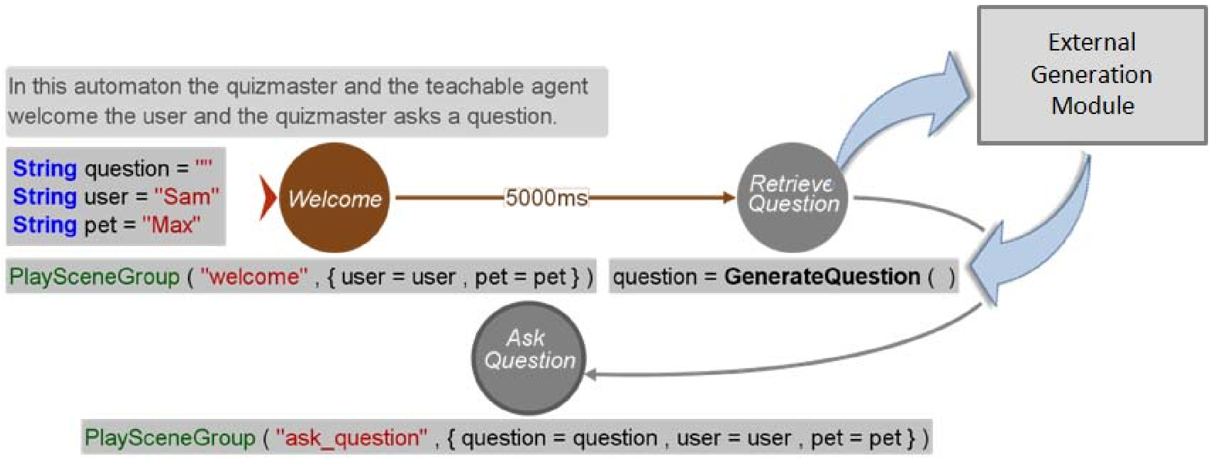
Furthermore, these state-transition diagrams do not provide adequate and convenient methods to react to the possible unanticipated turn-taking behavior of an interactive performance. An author does not have sufficient information about the past states of interaction, which is necessary to model special kinds of reopening strategies and recapitulation phases for dialogs and interactive performances. She or he has to collect that information by hand, which constitutes an enormous amount of modeling effort. However, adequate reopening strategies and recapitulation phases for dialogs and interactive performances are indispensable to give the user the impression of adaptiveness and flexibility of the virtual characters' interaction skills.

4.1.2. Concepts

The authoring suite *SceneMaker* [7, 10] makes use of a special kind of *hierarchical* and *parallel* state-transition diagrams, called *sceneflows*, which can be used for modeling parallel processes, as they occur in multi-party dialog and interaction with virtual characters. SceneMaker's visual programming environment for the creation of sceneflows allows the structured modeling of event-based interaction in a rapid and comfortable way, making time exposure and modeling effort scalable and appreciable. Sceneflows are easily extensible and reusable and they can be tested or debugged without much expenditure. For the realization of special reopening strategies for dialogs, SceneMaker provides a mechanism of *interaction history*. The interaction history provides an author with a convenient method to describe the possible unanticipated turn-taking behavior and the adequate response to potential interruptions of an interactive performance. SceneMaker facilitates the creation of interactive performances because it divides the authoring task into the creation of dialog content and the modeling of the narrative and structure of an interactive performance.

4.1.2.1. Organization of Dialog Content

Dialog content is organized in a set of parameterizable *scenes* that are specified in a multimodal *scenescript* which resembles a movie script with dialog utterances and stage directions for controlling gestures, postures and facial expressions. For each scene, there may be provided a number of variations, subsumed in a *scenegroup*, to increase variety and to avoid repetitive behavior that would impact a virtual character's believability. Scenes can be created manually by an author with standard text processing software. For this purpose, the SceneMaker authoring suite provides a *scenescript editor* with syntax highlighting and lexical analysis features. Furthermore, scene content can be generated automatically by external generation modules such as qualitative reasoning models or semantical phrase generation modules.


In this automaton the quizmaster and the teachable agent welcome the user and the quizmaster asks a question.
String question = ""
String user = "Sam"
String pet = "Max"
Welcome
5000ms
Retrieve Question
External Generation Module
PlaySceneGroup ("welcome" , { user = user , pet = pet })
question = GenerateQuestion ( )
Ask Question
PlaySceneGroup ("ask_question" , { question = question , user = user , pet = pet })

scene_en: welcome (1/3)

Q: Hi [joy] $user! I hope your teachable agent $pet has well prepared!

P: Hi [enthusiastic] $user. I'm [point] your teachable agent $pet.

scene_en: ask_question (2/5)

Q: Well $pet, next question: $question? Do you know the answer?

P: Hm, [think] let me think about that! May I have a cookie if I know the answer, $user?

Figure 3: Examples of calling scenes of the multimodal scenescript

The execution semantics of a scene depends on the actual character or game engine on which the scene has to be played back. SceneMaker's runtime application interface provides a *sceneplayer* interface which can be implemented by a concrete player that handles the actual playback of a scene.

Figure 4 shows a simplified diagram which shows the control and data flow between the sceneflow interpreter, the user-defined sceneplayer and the character or game engine: The sceneflow which was modeled in the SceneMaker editor is handed over to the *sceneflow interpreter*, which executes the sceneflow. Whenever a scene playback command in the sceneflow is executed, the respective sceneplayer is called. The sceneplayer makes a lookup into the scenescript to get the content of the scene taking into account possible gesticon or visicon data. The scene is then compiled into a sequence of actions or engine commands which are sent to the appropriate destination. After the scene has been played back the sceneflow interpreter continues with the execution of the remaining sceneflow.

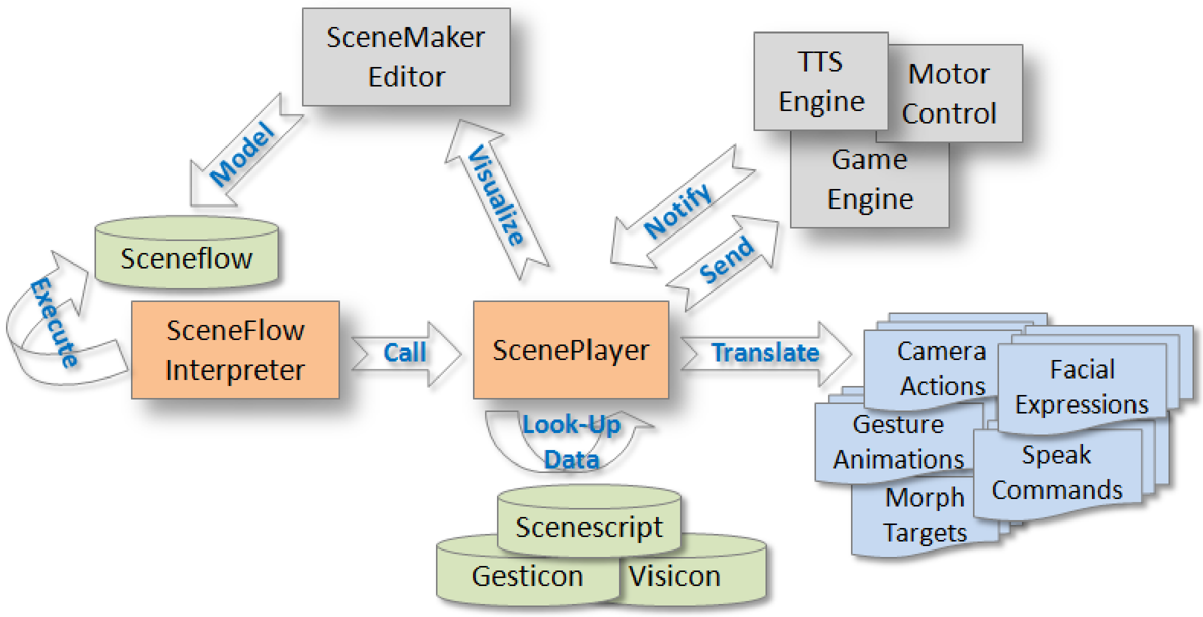


Figure 4: Simplified control and data flow during scene execution

4.1.2.2. Modeling Narrative Structure with Sceneflows

The narrative structure of an interactive performance is controlled by the sceneflow which is a hierarchical and parallel state-transition diagram specifying the logic organization and temporal order in which scenes are played or commands are executed. A scene flow consists of *scenenodes*, *supernodes*, different kinds of *edges* and a special kind of nodes, called *historynodes* resembling the system history. While each node can be linked to system commands or one or more scene playback commands, different branching strategies such as logical and temporal conditions as well as randomization can be used by specifying different edge between these nodes.

A supernode in a sceneflow may enfold several subgraphs, thus allowing for the *hierarchical refinement* and *parallel decomposition* of the sceneflow. This feature allows an author to carry out the modeling task in a divide-and-conquer manner. Except for synchronization measures, various control processes and user input processing as well as individual agents' behavior can be modeled independently of each other. Thus, the scene flow is kept clearly arranged, extensible and parts of it exchangeable.

The Syntax of Sceneflows: The following two types of nodes can be part of a sceneflow (see Fig. 8):

- Scenenode: Represents a state which allows the annotation with a command to play a Scene. A Scenenode may additionally be annotated with local type definitions, local variable definitions and variable assignments as well as with system commands like function calls to predefined functions of the underlying implementation language or native functions of the underlying operating system. Local type definitions, variable definitions and system commands are specified in a simple textual expressional language.
- Supernode: Extends the functionality of Scenenodes with features exploited to create the hierarchical structure as well as the parallel decomposition of sceneflows. A Supernode may contain both Scenenodes and Supernodes that together constitute its subautomata. At least one of these Subnodes has to be declared the Startnode of that Supernode. If a Supernode contains multiple parallel subautomata then each of these subautomata defines exactly one Startnode. In contrast to Harel's conventional statecharts, sceneflows allow multiple parallel automata to share common reusable subcomponents.

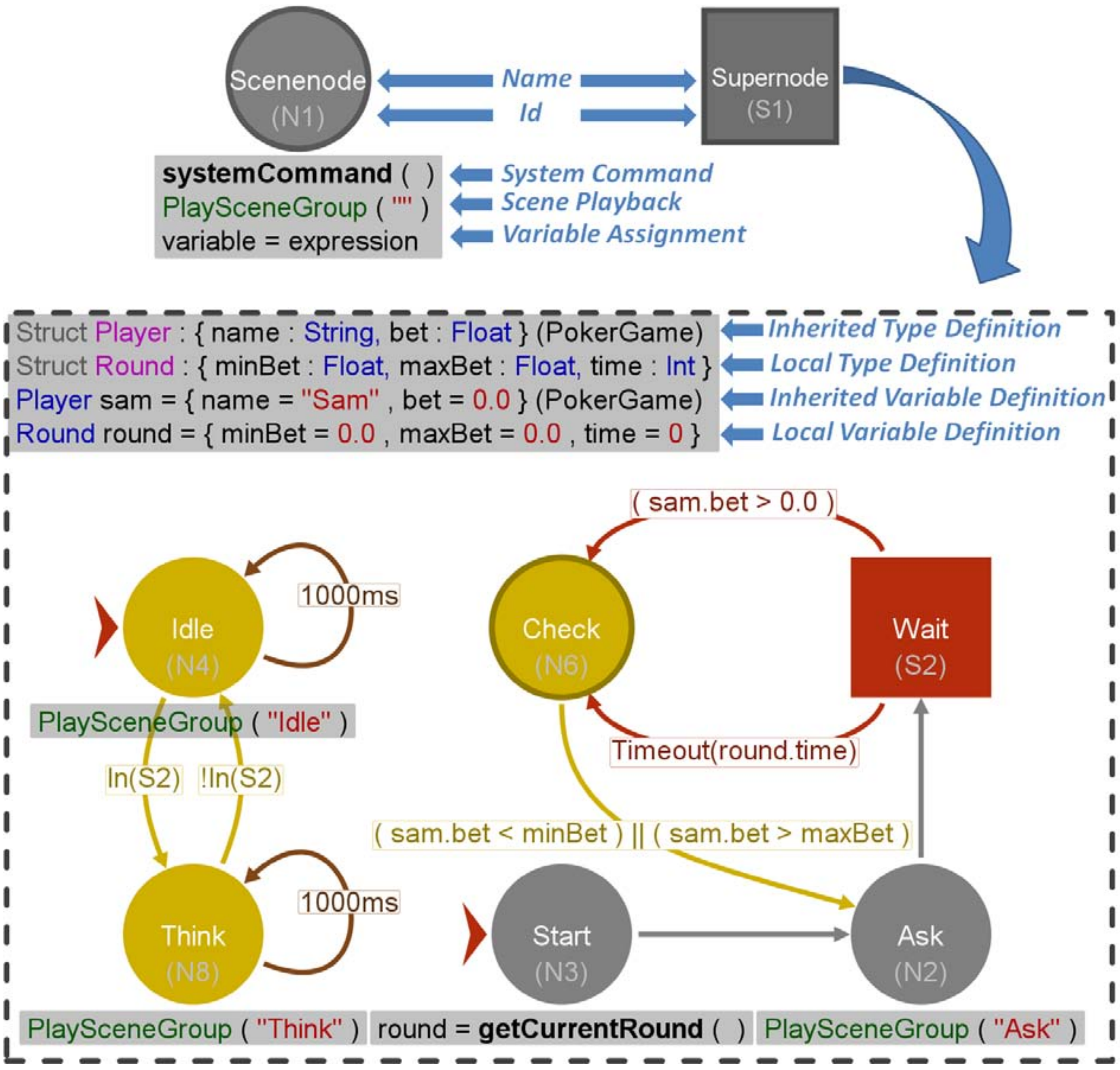

Scenenode (N1)
Name
Id
Supernode (S1)
systemCommand ( )
PlaySceneGroup ("")
variable = expression
System Command
Scene Playback
Variable Assignment
Struct Player : { name : String, bet : Float } (PokerGame)
Struct Round : { minBet : Float, maxBet : Float, time : Int }
Player sam = { name = "Sam" , bet = 0.0 } (PokerGame)
Round round = { minBet = 0.0 , maxBet = 0.0 , time = 0 }
Inherited Type Definition
Local Type Definition
Inherited Variable Definition
Local Variable Definition
(sam.bet > 0.0)
1000ms
Idle (N4)
Check (N6)
Wait (S2)
PlaySceneGroup ("Idle")
Timeout(round.time)
In(S2)
!In(S2)
(sam.bet < minBet) || (sam.bet > maxBet)
1000ms
Think (N8)
Start (N3)
Ask (N2)
PlaySceneGroup ("Think")
round = getCurrentRound ( )
PlaySceneGroup ("Ask")

Figure 5: Syntax of Scenenodes and Supernodes

The example in Figure 5 illustrates the use of parallel subautomata within a Supernode. The subautomata displayed in the dashed rectangle describe a simplified round of betting in a poker game. The right automaton models the behavior of the virtual character that is in the role of the dealer. At first, in node `Start` there is executed the system call `getCurrentRound` to the poker game engine to query the minimum and maximum permissible bet size as well as the time given to the user to make her bet. Next, the dealer invites the user to make her bet in node `Ask`. Thereupon, in Supernode `Wait`, the dealer waits until the user has made her bet or the time given is up. In node `Check`, the dealer checks if the bet is valid. If so, the betting round ends, otherwise the dealer invites the user to make a valid bet once again. The left subautomaton models the behavior of an avatar representing the user in a virtual poker room. In node `Think`, the avatar shows some thoughtful behavior while the user is thinking about her bet size. At any other time, it shows some variations of idle behavior. The points in time when the avatar changes its behavior are synchronized with the right automaton, so the dealer's behavior. The user has the possibility to communicate her bet over some user input interface which is not addressed in greater detail here, whereupon the avatar's behavior is indirectly controlled by the user's interaction over it's synchronization with the dealer.

An edge represents a transition between a source state and one ore more target states. Sceneflows define the following different types of edges:

- Epsilon Edge: Represents an unconditional transition (see Fig. 10(b)). Epsilon edges are used for the specification of the order in which computation steps are performed and Scenes are played back.
- Conditional Edge: Represents a conditional transition and is labelled with a *logical condition* (see Fig. 10(d, e, f)). The transition is taken when the logical condition comes true. Conditional edges are used to create a branching structure in the sceneflow which describes the different reactions to changes of environmental conditions.
- Timeout Edge: Represents a timed transition and is labelled with a *timeout value* (see Fig. 10(c)). The transition is taken when the timeout has expired. Timeout edges are used to regulate the temporal flow of a sceneflow's execution and to schedule the playback of Scenes and computation steps.
- Probabilistic Edge: Represents a transition that is taken with a certain probability and is labeled with a *probability value* (see Fig. 10(g)). Probabilistic edges are used to create some degree of randomness and desired non-determinism during the execution of a sceneflow.
- Fork Edge: A concept derived from the fork constructs of Harel's statecharts. Fork edges exploit the concept of parallel decomposition by allowing the splitting of a sceneflow's execution into several distinct processes that are executed *concurrently* (see Fig. 10(h)). Each of these concurrent processes executes a single *parallel subautomaton* of a sceneflow.

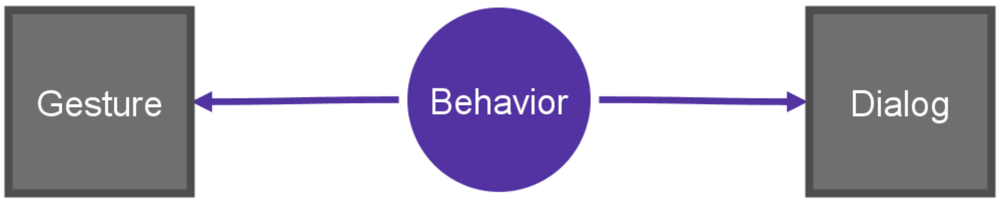


Figure 6: Concurrency with fork edges

Figue 6 shows how fork edges can be used for parallel decomposition when modeling the behavior of a virtual character. Two processes are created that concurrently execute the parallel automata Gesture and Dialog modeling the gestures and the dialog of the virtual character.

- Interruptive Edge: Is used for the handling of interruptive events requiring a fast reaction - usually events caused by user input or external events of the application environment. Interruptive edges have the task to *contemporarily* interrupt a running Supernode and all its subautomata as quickly as possible. They may be labeled with any kind of *condition* (see Fig. 7(i)).

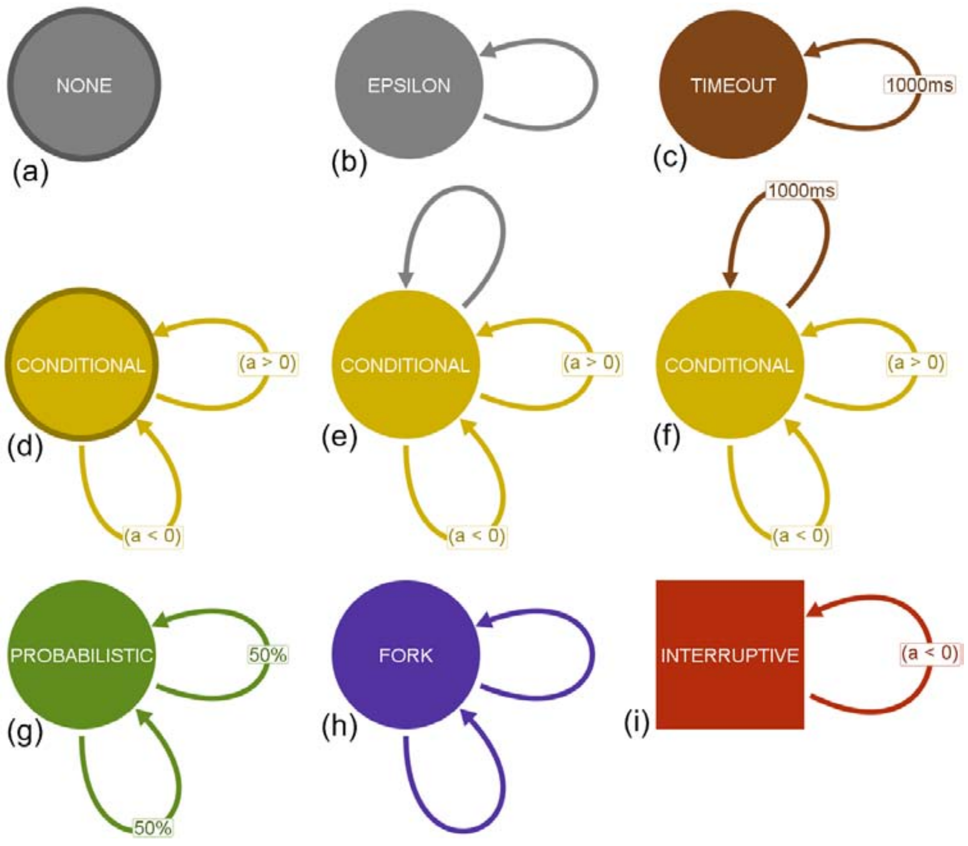


Figure 7: Different edge types and node flavours

The syntax of sceneflows allows only certain combinations of outgoing edge types of a node. These valid combinations lead to the definition of the different flavors that a node can have (see Fig. 7):

- None Node: A node that has no outgoing edges (see Fig. 7(a)).
- Epsilon Node: A node that has exactly one outgoing epsilon edge (see Fig. 7(b)). An epsilon node may not have more than one epsilon edge and cannot be end node.
- Timeout Node: A node that has exactly one outgoing timeout edge (see Fig. 7(c)). A timeout node may not have more than one timeout edge and cannot be end node.
- Conditional Node: A node that has outgoing conditional edges (see Fig. 7(d,e,f)). A conditional node may have an arbitrary number of outgoing conditional edges and if it has exclusively conditional edges then it is a possible end node (see Fig. 7(d)). A conditional node can have either one single epsilon edge (see Fig. 7(e)) or one single timeout edge (see Fig. 7(f)). If a conditional node has an epsilon edge or a timeout edge then it cannot be end node.
- Probabilistic Node: A node that has only outgoing probabilistic edges (see Fig. 7(g)). A probabilistic node may have an arbitrary number of probabilistic edges but no edges of other types. Also, it can not be end node. The probability values of all outgoing probabilistic edges have to sum up to 100.
- Fork Node: A node that has only outgoing fork edges (Fig. 7(h)). A fork node may have an arbitrary number of fork edges but no edges of other types and cannot be end node.

- Interruptive Node: A Supernode that has only outgoing interruptive edges (see Fig. 7(i)). An interruptive Supernode may have an arbitrary number of outgoing interruptive edges but no edges of other types and it is a possible end node.

Communication within Sceneflows: The individual behavioral functions and modalities that contribute to the behavior of a virtual character in an interactive performance are usually not completely independent, but have to be synchronized with each other. For example, speech is usually highly synchronized with non-verbal behavioral modalities like gestures and body postures. When modeling individual behavioral functions and modalities in separate parallel automata then the processes that concurrently execute these automata have to be synchronized by the author in order to coordinate all these behavioral aspects. Consequently, this requires a method for process communication in sceneflows. This communication is realized by a *shared memory model*. The syntax of sceneflows supports the following methods to synchronize concurrent processes.

- Shared Variable Access: One method for the communication between multiple concurrent processes is using shared variables that are defined in the variable scope of a common Supernode of those nodes that are currently executed by these processes.

- Local Variable Access: A second method for the communication between multiple concurrent processes is the direct access to a local variable of some node. For that purpose, sceneflows provide the expression `ValueOf(id,var)` which returns the current value of the variable `var` defined in the node `id`.

- System Configuration Query: This is a method which makes the synchronization and communication of concurrent processes more intuitive for an author. Sceneflows allow a query to the *system configuration* containing, at any point in time, the nodes that are executed by some process. For that purpose, sceneflows provide the condition `In(id)` which allows to request if the node `id` is currently executed by some process.

4.1.2.3. Using the Supernode History

The *history* of a Supernode facilitates modeling reopening strategies and recapitulation phases of dialogs and interactive performances by falling back on automatically gathered information on past states of interaction. The history concept of sceneflows enfolds the following components:

- History Memory The history of a Supernode automatically keeps track of the last substates the Supernode was in, values of data fields and the last system commands that were executed or Scenes that were played back. Sceneflows provide several expressions, conditions and commands to request these information and to manipulate or delete the history memory of a Supernode [10]. The automatically maintained history memory releases the author of the manual collection of these data, thus reducing the modeling effort.

- History Node When reentering a Supernode that had already been executed and afterwards interrupted in the past, then the Supernode's subautomat starts at the *Historynode* instead of the Startnodes of this Supernode. Thus, the Historynode serves as orientation and starting point for the author if he or she models reopening strategies for the case of a reenacted execution of a Supernode.

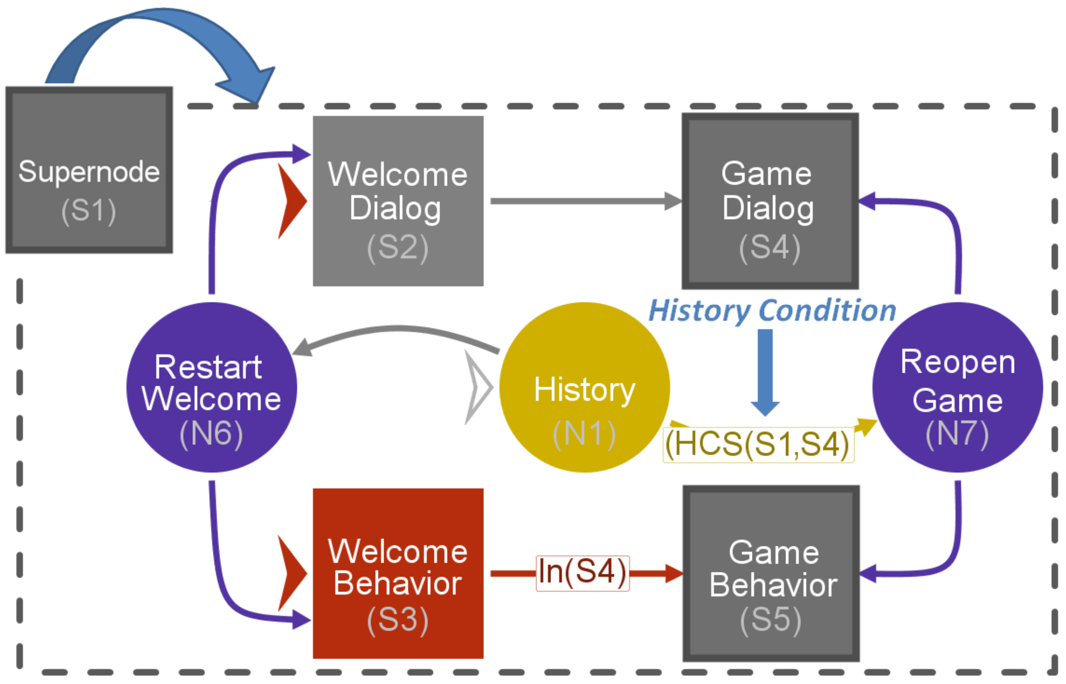


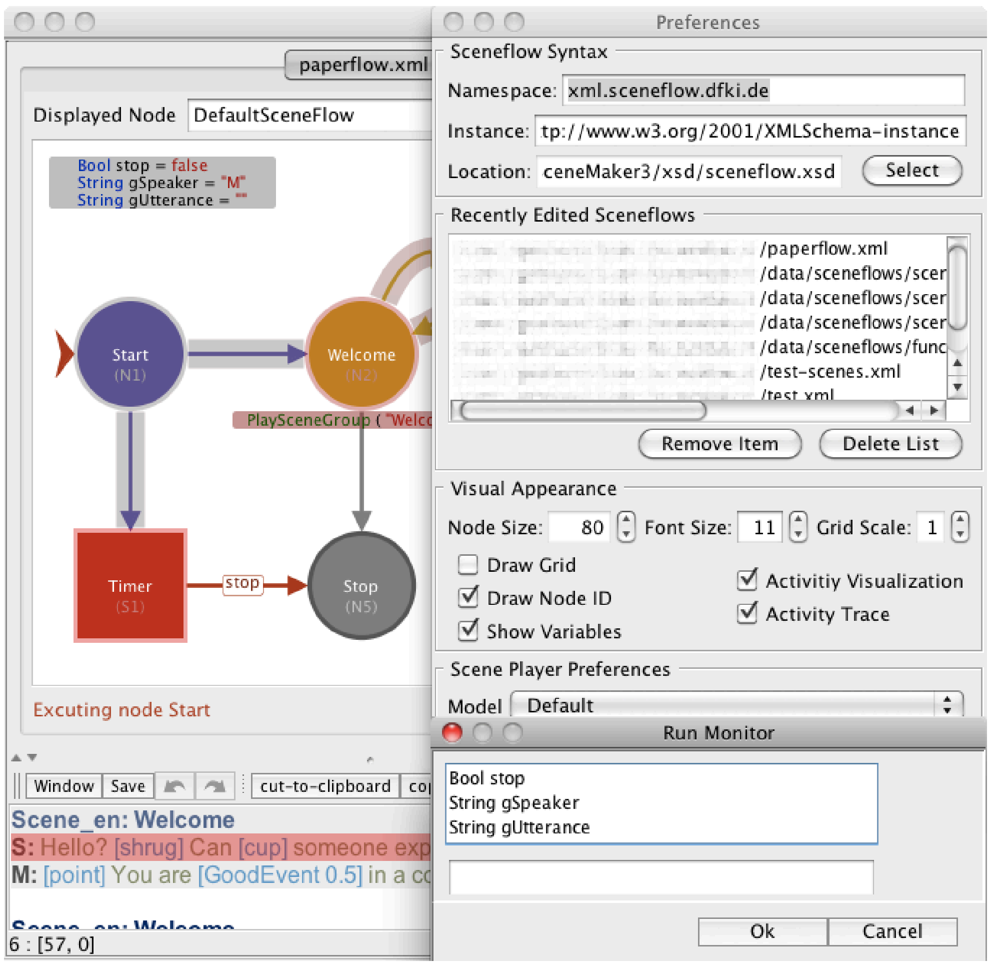
Figure 8: Historynode and Historyconditions

Figure 8 shows an exemplary use of the Historynode and the recorded history memory information. It shows the two parallel subautomata of a Supernode `S1`. The lower subautomaton models the non-verbal behavior of a virtual character while the upper subautomaton models the synchronized dialog behavior. `S1` can be divided into two phases: First, the welcome phase in which the Supernodes `Welcome Behavior` and `Welcome Dialog` model the welcome behavior and the welcome dialog. Second, the game phase in which the Supernodes `Game Behavior` and `Game Dialog` model the game behavior and dialog. If `S1` is interrupted at some time and afterwards reentered then it starts at the Historynode `History`. Starting from the Historynode, the history memory is requested by using the Historycondition `HCS(S1,S4)` to find out if `S1` had been interrupted while executing Supernode `Game Dialog` which means that it happened during the game phase. If not, then `S1` had been interrupted in the welcome phase and so the welcome phase is restarted over the node `Restart Welcome`, otherwise the game phase is reopened over the node `Reopen Game` without replaying the welcome phase once again.

4.1.3. Visualization Software

The visualization software is supposed to allow an author to test, simulate and debug a model in an early phase of development. It helps the author to control the modeling progress and to verify the correctness of the model. The visualization is based on the graphical user interface of the SceneMaker authoring tool. Figure 9 shows a screenshot of the SceneMaker authoring tool during the execution of a sceneflow. On the upper right side there is displayed an option dialog that allows changing a variety of visualization settings. There may be selected a *runtime visualization mode* which causes a temporary highlighting of executed nodes, edges and Scenes in a sceneflow. Additionally, there may be selected a *trace visualization mode* which causes the permanent highlighting of paths that have been taken in a sceneflow. On the top left side there can be seen the workspace of the editor which contains the sceneflow and a variable badge of the currently displayed Supernode. There can be seen that nodes, edges and Scenes are highlighted when executed and that the variable badge always shows the actual value of local and inherited variables. The bottom shows an integrated text editor which may be used to edit the Scenes. When a Scene is executed in a sceneflow then this Scene and the contained utterances

are also highlighted within the text editor. Finally, on the right side at the bottom there is shown the runtime monitor of SceneMaker. The runtime monitor shows the global variables of a sceneflow and allows the assignment of these variables with arbitrary expressions in the textual expression language that are evaluated at runtime. The runtime monitor may be used to simulate any kind of user input or sensor input during the execution of a sceneflow.


paperflow.xml
Displayed Node DefaultSceneFlow
Bool stop = false
String gSpeaker = "M"
String gUtterance = ""
Excuting node Start
Scene_en: Welcome
S: Hello? [shrug] Can [cup] someone exp
M: [point] You are [GoodEvent 0.5] in a co
6 : [57, 0]

Preferences
Sceneflow Syntax
Namespace: xml.sceneflow.dfki.de
Instance: tp://www.w3.org/2001/XMLSchema-instance
Location: ceneMaker3/xsd/sceneflow.xsd
Recently Edited Sceneflows
/paperflow.xml
/data/sceneflows/scer
/data/sceneflows/scer
/data/sceneflows/scer
/data/sceneflows/func
/test-scenes.xml
/test.xml
Visual Appearance
Node Size: 80 Font Size: 11 Grid Scale: 1
☐ Draw Grid
☑ Draw Node ID
☑ Show Variables
☑ Activitiy Visualization
☑ Activity Trace
Scene Player Preferences
Model Default

Run Monitor
Bool stop
String gSpeaker
String gUtterance

Figure 9: The SceneMaker authoring tool

4.2. Using SceneMaker in DynaLearn

As an example of how to realize and use the scaffolding principle in DynaLearn we prototypical implemented the *Diagnosis* use case, using a mockup of the diagnosis itself. In this use case a virtual character (or several) evaluates the model created by the learner and helps them to correct it in a stepwise manner.

For this we combine the elaboration of Lipscomb et al. with the dialog modes of Cade et al. The more widespread scaffolding of Lipscomb et al. determines the general behavior of our teacher, while we use dialog modes of Cade et al. as dialog steps within this behavior.

As its overall task, the virtual character analyzes the model for mistakes and flaws and tries to lead the learner to the solution of the current problem step-by-step. For this purpose the "real" teacher provides different teaching aids for the learner from one of the following three categories: Lecture, Scaffolding and Modeling. The character usually chooses with an equal chance between lecture and scaffolding when providing an aid. These chances are based on the observations of Cade et al. where lectures and scaffolding were the most present dialog modes with a very similar frequency. As we ideally want learners to find the solution by themselves, modeling (i.e. the exact correction of the mistake) is only appropriate if the character has exhausted all other means. In this way, we ensure learners can proceed with the correction of their model even if they cannot cope with a particular problem.

The next section will describe the technical details of this behavior.

4.2.1. Commands

First we have to describe how the computation of guidance works. Since the dialog management is only responsible for behavior and sound dialogs we assume that the QR component does the analysis of the learner's model and provides the means the virtual tutor can use to help. Except lectures that can be prepared in scenes because their content does not change with a specific problem. It is just necessary to tell the dialog management which lecture fits the current problem. In principle we assume that the QR component provides a pool of aids that could help the learner to find the solution to the current problem. The dialog management chooses from this pool and integrates the aid into an ongoing dialog.

We implemented several special commands in SceneMaker for this:

`requestTopic()` - most problems can be assigned to a general topic like "entities and quantities". The correction of a model is the result of processing all topics that are not flawless in the model. With this function, the character requests a new topic that contains a general error message for the learner conveyed by the agent, like "There seems to be something wrong with your entities and quantities". Also the pool of resources the character can use to help the learner find the exact mistakes and solutions for them.

`getTeachingModes()` - this command delivers according to the current probability distribution an integer (1 to 4) that represents a specific category of dialog modes. For example 1 means that the character should use one of the fitting lectures to help the student. If the chosen teaching mode is 4 the character has exhausted all means (even modeling) and can not help the student any more.

`checkResults()` - this command tells the QR component to check the learner’s model again. This command is used after the learner tells the character that they are done with the correction of the model. With this the dialog management checks whether the topic is really solved. If not the agent has to continue his guidance for this topic.

`changeLikelihood(chosenMode, value)` - as the specific help is chosen by chance it is self-evident to change this likelihoods according to the success of the last used aid. For example the chances for scaffolding should rise significantly when lecture does not seem to lead the learner to the solution.

4.2.2. Tags

The scenes for the characters also include placeholders and instructions for the visualization component. The necessary information is saved in tags:

$ID - these tags are placeholders with ID as an identifier that is matched and replaced with the corresponding parameter of PlaySceneGroup()

[ClipName] - these tags signalize an animation that should be played with the corresponding utterance. This tag is also deleted during the processing of the scene.

4.2.3. Implemented Tutoring / Communication Strategies

Following the idea of scaffolding our dialog management helps the learner to correct his model by guiding him through all flawed parts of his model. The agent retrieves one topic after another starting with fundamental issues heading towards more specific problems (see figure 10).

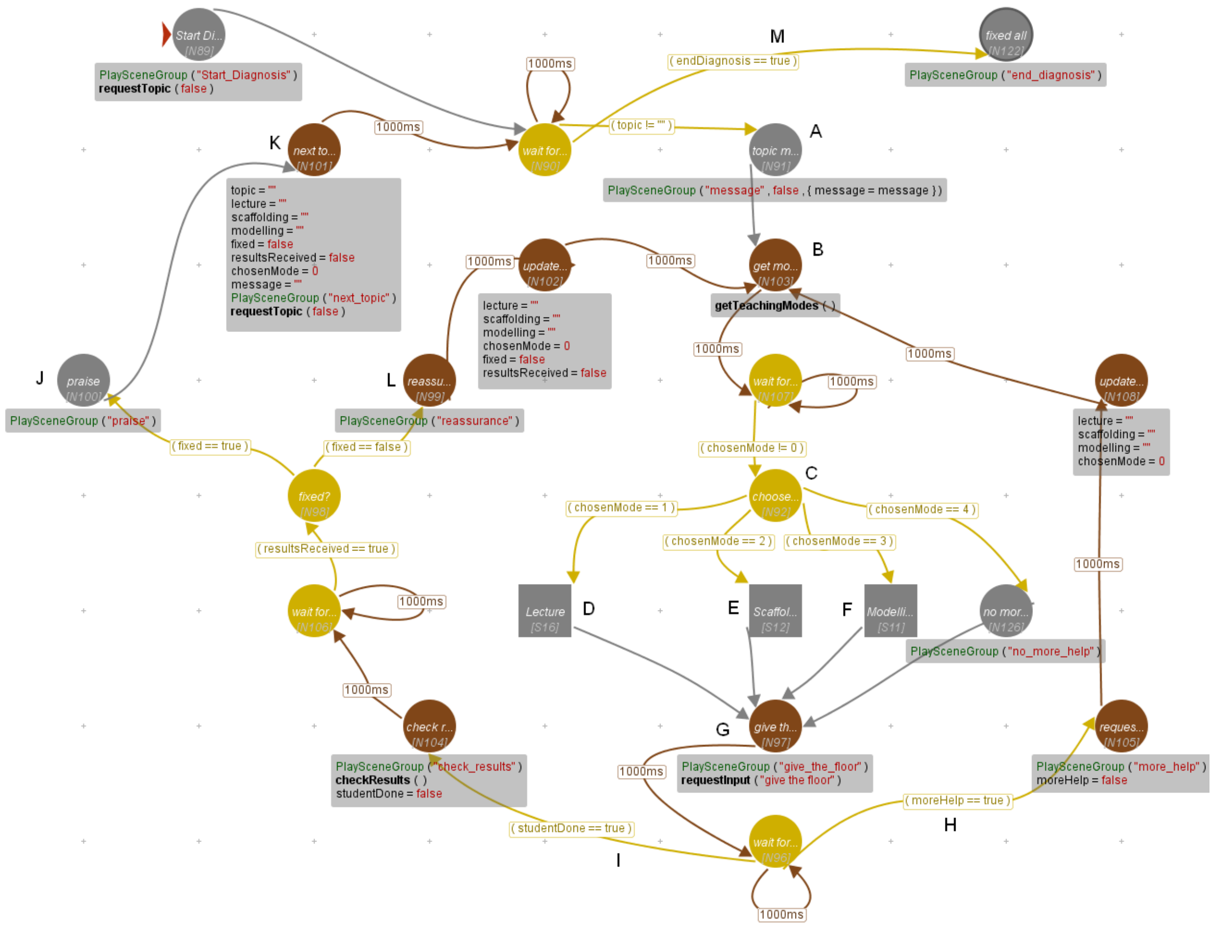


Figure 10: The state machine implementing scaffolding

The character starts each cycle (the processing of one topic) with a general error message (marked "A" in Figure 10). This roughly tells the learner which part of the model is not correct but they still have to find the specific mistakes by themselves. The character then requests the available teaching modes for this topic (B) and chooses one of them (C).

Depending on this choice, one of the supernodes (D, E or F) is entered. Within them, the logic for the specific dialog mode is handled. For example within the *Lecture* supernode (see Figure 11) the agent asks after his lecture if the learner has understood ("A" in Figure 11) and waits for input (B). If the learner has not understood (C), the agent can provide - if available - a more detailed version of this lecture (D). This goes on until no more details are available (E) or the learner has understood (F). In any case, the lecture is now over (G).

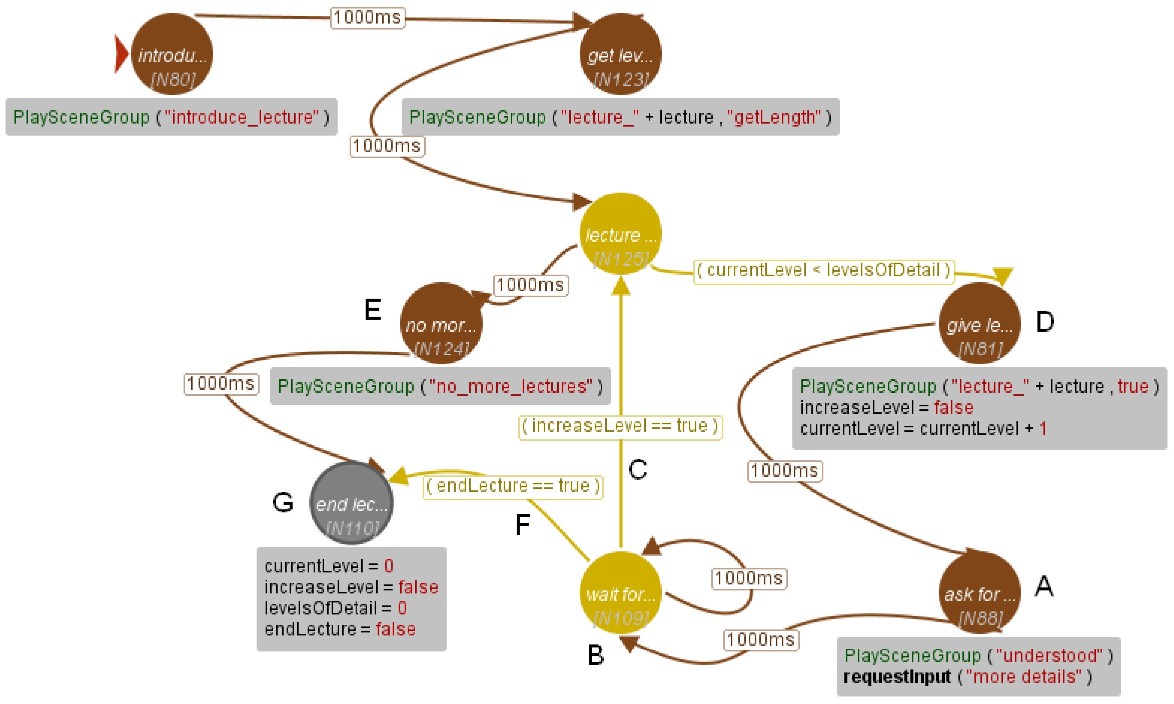

introdu... [N80]
PlaySceneGroup ("introduce_lecture")
1000ms
get lev... [N123]
PlaySceneGroup ("lecture_" + lecture , "getLength")
1000ms
lecture ... [N125]
(currentLevel < levelsOfDetail)
E
no mor... [N124]
1000ms
1000ms
PlaySceneGroup ("no_more_lectures")
give le... [N81]
D
PlaySceneGroup ("lecture_" + lecture , true)
increaseLevel = false
currentLevel = currentLevel + 1
(increaseLevel == true)
C
1000ms
G
end lec... [N110]
(endLecture == true)
F
currentLevel = 0
increaseLevel = false
levelsOfDetail = 0
endLecture = false
wait for... [N109]
1000ms
1000ms
ask for ... [N88]
A
B
PlaySceneGroup ("understood")
requestInput ("more details")

Figure 11: Interior of the supernode *Lecture*

In the supernode for *Scaffolding* we see which means the character can provide with this dialog mode (see Figure 12).

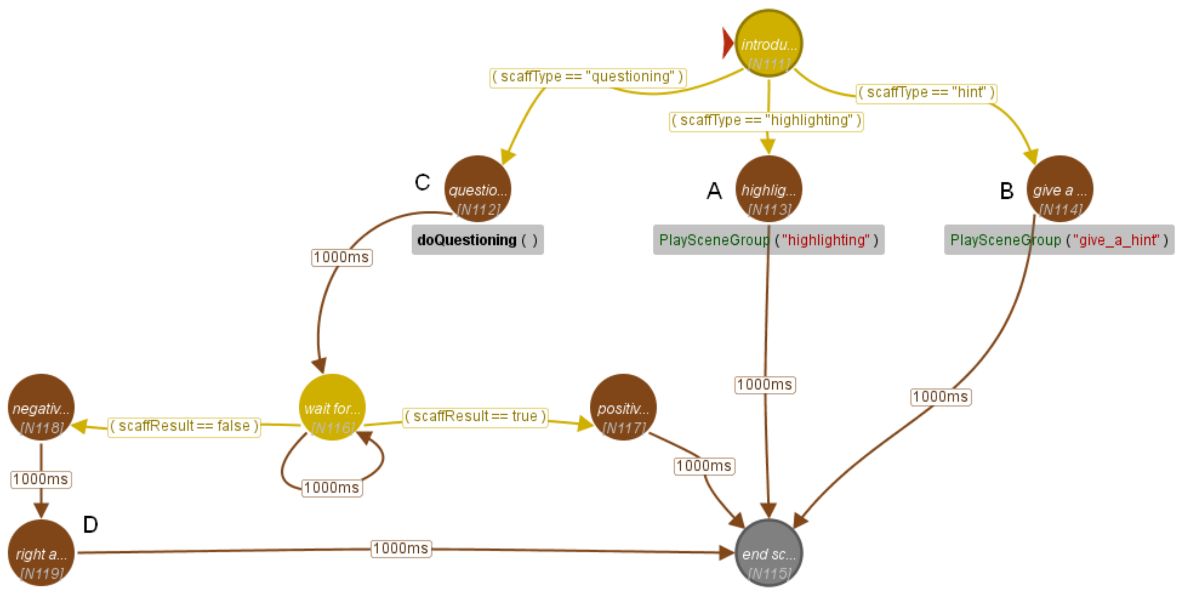


Figure 12: The interior of the *Scaffolding* supernode

Hints and *Highlighting* are quite similar. *Highlighting* ("A" in Figure 12) uses simple statements to draw learners' attention to specific parts of the model (see Figure 13 for an example). *Hints* (B) are more to the point and indicate what the problem might be (see Figure 14 for an example). *Questioning* (C) involves learners a bit more directly by asking them to decide which option is right and correcting them if necessary (D). See Figure 15 for an example of questioning.

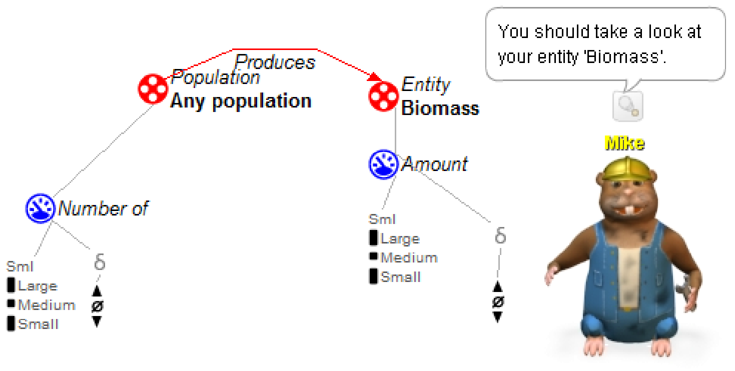


Figure 13: Example of Highlighting

In Figure 13, the character uses highlighting to point to the fact that the entity "Biomass" should rather be modeled as a quantity.

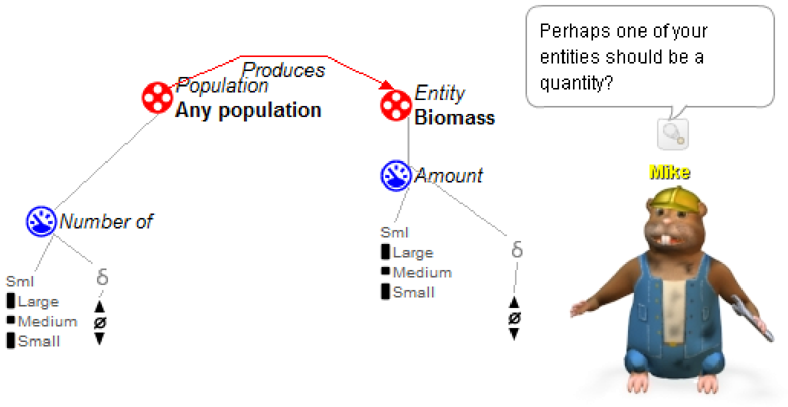


Figure 14: Example of a Hint

In Figure 14 we have the same situation as above. This time, however, the character gives a hint.

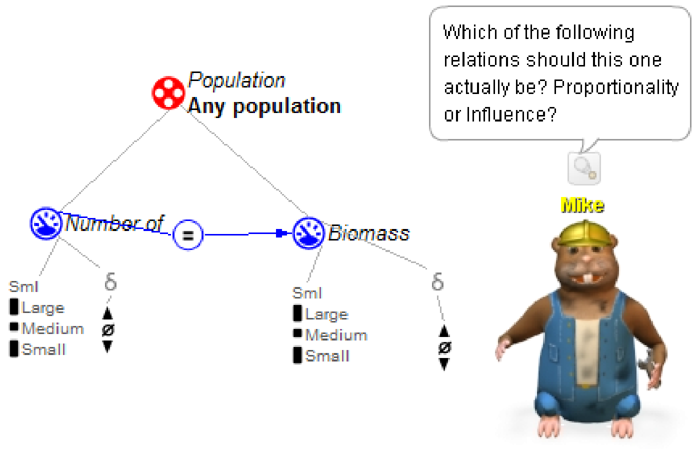

Population
Any population
Number of
=
Biomass
Sml
Large
Medium
Small
δ
Which of the following relations should this one actually be? Proportionality or Influence?
Mike

Figure 15: Example of Questioning

In Figure 15, the character uses questioning to point the learner towards the fact that the equality should actually be a proportionality.

After delivering the aid, the character now asks the learner to try and solve the problem ("G" in Figure 10). From this point on, learners can always ask for more help, which results in a new choice of a teaching

mode (H), or declare they are done with the correction (I). If this has really fixed the error, the character praises the learner (J) and goes on with the next topic (K). Otherwise he tells the learner that he is unfortunately not done (L) and gives him a new advice.

This goes on until all topics are processed and hence the model is corrected (M). During this process every step is accompanied by an utterance of the agent. In the resulting dialog, learners always know what the agent expects from them and that the system is reacting to their interaction.

5. Verbalization

As mentioned above, the process of Verbalization deals with the exact choice of words once the content that is to be communicated is decided.

For the verbalization of the virtual characters' utterances in DynaLearn, we investigated the following concepts and approaches:

- Verbalization of ontologies, e.g. Natural OWL [13]
- Natural language realisations engines, e.g. SimpleNLG [14]
- XML-based Natural Language Generation [15]

However, since these approaches were either not suited for a multiple languages or introduced too much overhead in an implementation, we decided to use SceneMaker's concepts of scenegroups and placeholders instead.

Basically, scenes with placeholders can be seen as templates which are then filled with appropriate data. Figure 16 shows two different scenes from the same group which can be used to announce quiz results of their teachable agent (see Deliverable 5.3. for details) to the learner.

```
Scene_en: announce_results
Q: Well, $user, your teachable agents $TA achieved $score points today!

Scene_en: announce_results
Q: Congratulations, $user! In the last quiz, $TA got a score of $score!
```

Figure 16: Two example scenes with placeholders

In this case, the quizmaster ('Q') character would randomly choose one of the two scenes and the placeholders could be filled with the values of the respective variables.

In case of communicating conceptual knowledge related to a specific QR model, such as questions or explanations, the actual data has to be extracted first. As an example let us take a look at the answer to a What is?-Question from the Basic Help use case (again, see Deliverable 5.3. for details). Let us assume that a learner is looking at the model depicted in Figure 17.

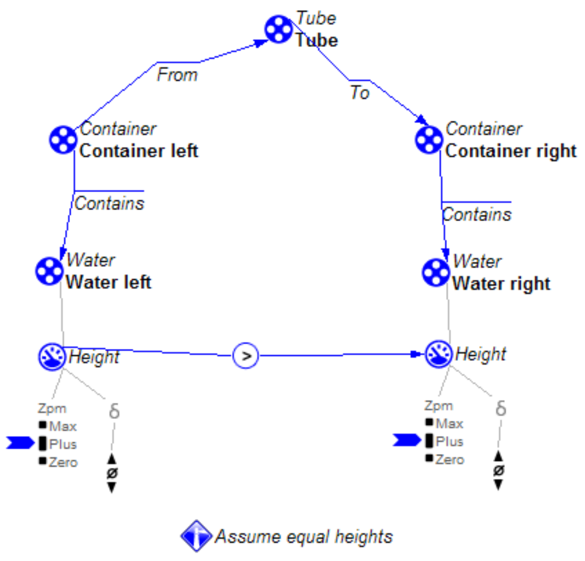


Figure 17: Communicating Vessels example model

If the learner were now to ask 'What is Water right?', the RDF-representation listed in Figure 18 would then be send to the IM from the CM.

```
<rdf:RDF xml:base="http://www.dynalearn.eu/models/LS6-CommunicatingVessels">
  <rdf:Description rdf:about="entityInstance#Water_right_000000040">
    <qr:hasName xml:lang="en" rdf:datatype="http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#string">Water right</qr:hasName>
    <qr:hasRemarks xml:lang="en" rdf:datatype="http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#string"/>
    <qr:hasCategory xml:lang="en" rdf:datatype="http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#string">entityInstance</qr:hasCategory>
    <qr:hasState xml:lang="en" rdf:datatype="http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/#string">consequence</qr:hasState>
    <qr:inAggregate rdf:resource="scenario#Positive_but_left_greater_than_right_000000020" name="Positive but left greater than right"/>
    <qr:hasDefinition rdf:resource="entityDefinition#Water_000000002" name="Water"/>
    <qr:hasQuantityInstance rdf:resource="quantityInstance#Height_000000049" name="Height"/>
    <qr:hasConfiguration rdf:resource="configurationInstance#Contains_000000044" name="Contains" otherName="Container right"
    otherID="entityInstance#Container_right_000000042" direction="fromOtherToThis"/>
  </rdf:Description>
</rdf:RDF>
```

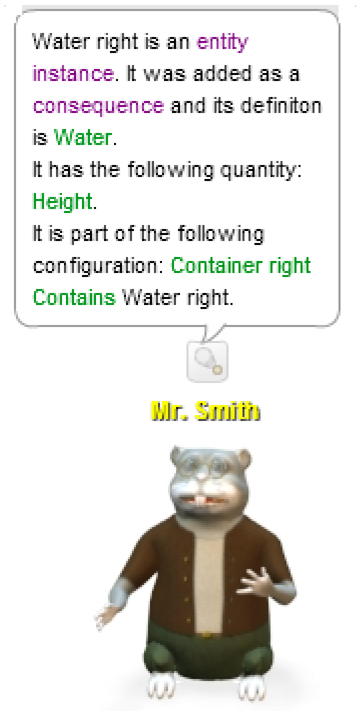
Figure 18: Example of an answer represented in RDF

The representation is then parsed, using the OWL API [16]. This gives us the necessary data to fill in the placeholders in the scene file shown in Figure 19.

```
Scene_en: what_is_entity
T: $EntityName is an entity.
   It was added as a $EntityState and its definition is $EntityDefinition.
   It has the following quantity(ies): $ListOfQuantities.
   It is part of the following configuration(s): $ListOfConfigurations.
```

Figure 19: Example scene for explaining an entity

For the sake of completeness, Figure 20 shows how the teacher character then presents the explanation.



Water right is an entity instance. It was added as a consequence and its definiton is Water.
It has the following quantity: Height.
It is part of the following configuration: Container right Contains Water right.

Figure 20: The teacher character explaining an entity

Please note that in this case, the terms "quantity(ies)" and "configuration(s)" are replaced with the correct ones, once the number of items in the respective list is known and before the utterance is sent to the virtual characters.

6. User Model

Assessing the knowledge of a student is a fundamental part of intelligent learning environments. We employ a Bayesian Network (BN) based approach to dealing with uncertainty when estimating a learner's state of knowledge in the context of Qualitative Reasoning (QR). A proposal for a global architecture is given. The essentials of the belief network structure for individual scenarios are described, while paying special attention to knowledge aggregation and some design issues that are specific for the domain of QR.

6.1. Bayesian networks as models of learners

Probabilistic graphical models, diagrammatic representations of probability distributions, are useful in analyzing and solving complicated probabilistic models. They consist of nodes connected by edges, where each node represents a random variable, and each edge denotes a probabilistic relationship between the variables it connects. BNs are represented in terms of a directed graph. An important restriction that applies to such directed graphs is that they must be kept *acyclic*. In other words, there must be no closed paths within the graph that would allow us to start at one node, follow a path and come back to the point of origin. Therefore, BNs are essentially *directed acyclic graphs* (DAGs).

Knowledge tracing was an early approach to student modeling that exploited the power of Bayesian inference, initially used by Corbett and Anderson in their ACT Programming Tutor [23], but was also proven to be useful for designing tutors for mathematics [25], and improving reading skills [18], and is statistically equivalent to Reye's two-node dynamic BN used in many other learning environments [27]. Corbett and Anderson proposed a two-state model: each topic is either *learned* or *unlearned* (i.e. known or unknown). The model does not implement the idea of *forgetting*, but it does take the possibility of *guesses* and *slips* into consideration.

Their knowledge tracing approach uses four initial parameters:

- G – the guess parameter, i.e. the probability of the student's correct answer being a 'lucky guess' (when they don't know the actual correct answer);
- S$ – the slip parameter, i.e. the probability of the student accidentally giving an incorrect answer (when they actually know the correct one);
- L_0 – the probability of knowing a skill at the beginning of a tutoring session;
- T – the probability of learning an 'unknown' skill, at any given point during a tutoring session.

Therefore, the probability of knowing a certain topic after an action *n* is calculated using that action as evidence for the posterior probability (see equation below). In the APT, the student is given exercises until their mastery level reached the probability of 95%.

$$\mathrm{p}(L_n|evidence) = \mathrm{p}(L_{n-1}|evidence) + ((1-\mathrm{p}(L_{n-1}|evidence))p(\mathrm{T}))$$

Beck [19] pointed out a deficiency of most methods for developing Bayesian Knowledge Tracing models for specific skills – the *identifiability* problem, where models with equally good statistical fit to performance data may make very different predictions about a learner's knowledge state, which could result in different numbers of problems to be assigned to a student. Consequently, the problem could result in under- or over-practice.

Baker et al. [17] proposed a new method for finding two of the four basic parameters, that differs in estimating the guess and slip parameters for individual actions (i.e. related to the context), instead of holding them constant for all situations.

6.2. Bayesian learner model for DynaLearn

In DynaLearn, the information flow starts with the user who constructs and simulates QR models. The information from these models and simulations is used to construct a BN. Based on the information in the network, i.e. the probabilities of knowing each concept, a question focus is determined, and a question request is sent to a QUAGS (QUestions about Garp Simulations), a question generator [24]. The question generator uses this request, the information about the models, question templates and a number of criteria to output a list of possible questions. Then, a question is selected and forwarded to the learner. For more information on the question generator process, see Deliverable 3.3. After the learner provides an answer, the information about the question and the response is stored in the dialog history, and the belief network is updated.

In Brielmann's [22] learner model, each primitive is represented as a node in the model BN, where magnitudes, derivatives, dependencies and correspondences are set to be the root nodes. We have already discussed the knowledge aggregation methods proposed by Reye [27] and Millán et al. [26], and Brielmann adopts the same approach in her model. We can observe the low level concepts as the concept nodes, the quantities as topic nodes, and entities as subject nodes found in [26]. That is, in order for a student to understand an entity, they must know all of its quantities first. In order to know a quantity, on the other hand, a learner must understand all the root concepts directly related to it. The idea is shown in Figure 21 for a simple 'Tree and shade' model [20]. It is supposed to describe the relation between a growing tree and the shade that is being cast on the ground by the tree. We see that the proportionality and influence nodes both contribute to two upper level nodes, as each of them carries knowledge about two concepts.

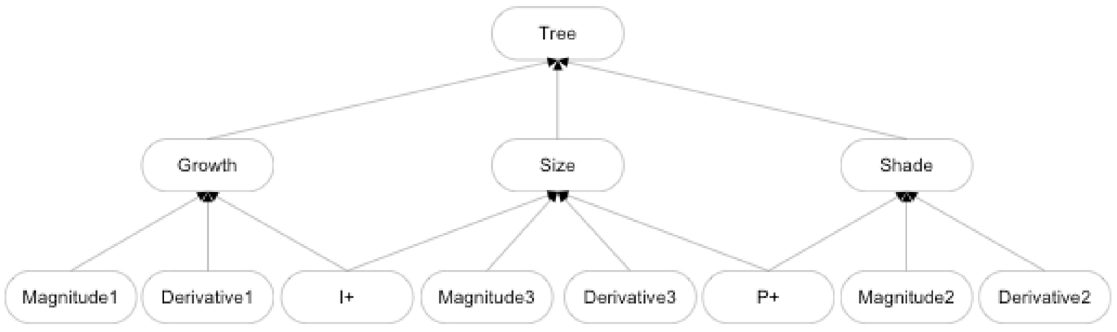


Figure 21: Tree and shade: Knowledge aggregation

Each root node is connected to an extra child node representing an observation from a system-learner interaction. This approach makes it possible to include the possibility of guesses and slips, as proposed by [27]. As the learner's knowledge changes over time, after each interaction, the network is extended by another time slice. This lets us collect more than one piece of evidence for a piece of knowledge and provides us with the means to add the possibility of the learner learning or forgetting something between two points in time/interactions, as discussed in the earlier sections.

A number of key issues are not covered by [22]. If a model has multiple scenarios, the system would see each scenario as a completely separate domain. Moreover, it does not cover recurring concepts or substructures within the same scenario. Learning about one instance of a MF should imply we have also learned something about any other instance of the same MF.

6.3. Global architecture

Before we elaborate on the generation of the Bayesian Network, let us first take a look at the architectural considerations of our learner model.

A single scenario (and simulation) within a QR model holds only partial information about the domain covered by the model. Similarly, a single QR model could be only a small part of a bigger domain a student's supposed to cover. Therefore, our learner model should not only be able to handle individual scenarios, as per single examination session provided by the question generator, but also support moving between scenarios, or even models.

Two possibilities seem appropriate. We could either update the information inside the existing BN, or create a new (global) one, based on the information contained in the previously created network(s). We know, however, that we do not want to simply move the knowledge directly from one scenario to another and aggregate unnecessary information as we move on. Also, trapping the acquired information in a single external network is definitely not the way to move forward. Therefore, an ideal approach should store acquired knowledge in an external repository after each examination session, but also be able to retrieve the necessary information from that source when switching to a new scenario.

When loading a new simulation, the system would compare the concepts occurring in it with the ones contained in the model knowledge base. It would then create the belief network for the given scenario and update the information about the concepts the student has already seen earlier accordingly. The same approach could be easily extended to the global/domain level, so that the information would flow between a domain knowledge base (KB) and the individual scenario KBs.

6.4. Individual scenario with simulation

What needs to be learned? We should pay special attention to the low level (system) view, as it holds the essential information stored in each QR model, as suggested by [22]. To determine exactly what low-level concepts need to be represented in the network structure, we need to see first what questions can be asked, and what information the answers to questions hold.

An analysis of all QUAGS question types has produced the following list of low level concepts (to be explained below) the answers to such questions can hold information about: (1) Magnitudes, (2) Derivatives, (3) Quantity spaces, (4) Influences and proportionalities, (5) Inequalities, (6) Calculations, and (7) Correspondences. Each of the items in this list can be addressed directly by a QUAGS question. We consider quantities, entities and MFs high-level concepts, and the question generator does not ask any direct questions about any of them. Keep in mind, though, that if the current focus of the question generator was a certain quantity, we would like to track the student's knowledge about that quantity. Therefore, we are not saying high-level nodes should be left out from the network structure.

6.4.1. Recurring concepts

Imagine the following setup – in an additional MF of the Tree and shade model, instead of one, we have two trees, where one grows in the shade of the other (bigger) one. Therefore, the more the big tree grows, the slower will the small tree grow. Keep in mind that this third MF would become active only if the scenario setup allowed for it. Now, suppose in one scenario, this new MF activates. In terms of low-level concepts seen above, same forces are at work for both trees. This is due to the fact that they are merely instances of the same entity, that serves as a condition for two other MFs, that specify the dynamics of the tree and shade growth process. Going back to the original question, i.e. "What needs to be learned?", we realize that what we want to learn about is the Tree entity, and any instances of this entity should be treated as such. That is, every instance of a concept carries a certain amount of information about the generic entity, but the global (model/domain) knowledge base should keep track only of the generic concepts, as this is what we want to know about.

6.4.2. Recurring substructures

The example with two trees given above served another purpose by bringing up the following question: if a student learns something about one of the trees, have they not learned something about the other one as well? The problem at hand is a peculiar one, as it suggests the information in such cases should flow both ways. We already know that loops are not allowed in BNs. That means that a direct connection from one instance to another, as shown in Figure 22, would not work. Another approach, preserving the guess and slip factor for instance-specific question nodes, would also result in an illegal BN structure (Figure 23). Many other options also fail.

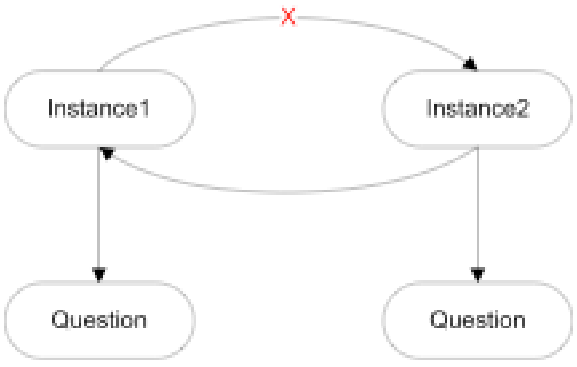


Figure 22: Illegal graph - Direct connection

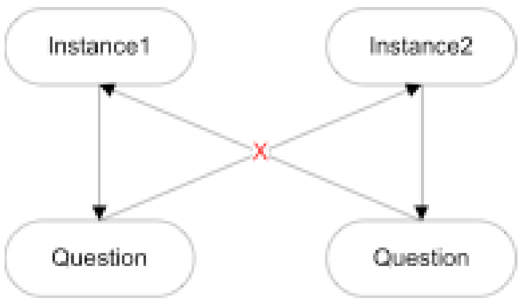


Figure 23: Illegal graph - Via question nodes

Therefore, we avoid the cycle trap through the idea of answer collectors. This approach adds an additional layer to the network, between the low level concepts and the question (outcome) nodes. The structure in Figure 24 shows a low level concept called *PositiveProportionality AH* and three of its instances. Each answer collector, as its name implies, collects answers coming from the outcome nodes and distributes it among all of the instance nodes. Moreover, the idea of guesses and slips is preserved.

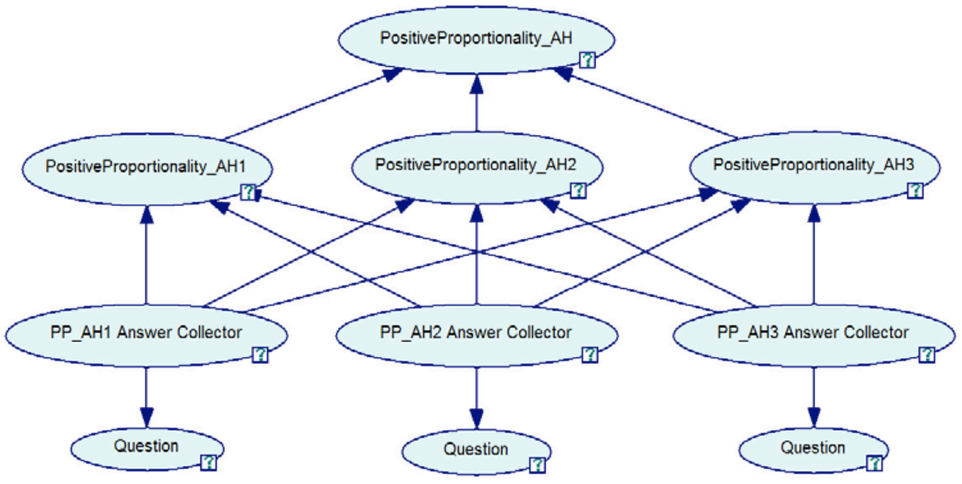

PositiveProportionality_AH
PositiveProportionality_AH1
PositiveProportionality_AH2
PositiveProportionality_AH3
PP_AH1 Answer Collector
PP_AH2 Answer Collector
PP_AH3 Answer Collector
Question
Question
Question

Figure 24: Recurring concept knowledge distribution via answer collectors

6.4.3. Soft evidential updating

As [28] explain, evidence is a collection of hard or soft findings on variables. Whereas a hard finding specifies which value a variable is in, a soft one specifies a probability distribution of a variable. Soft evidence is a collection of soft findings. In other words, instead of saying a concept is simply known or unknown, we could directly involve uncertainty using soft evidence and, after an interaction, say something is, for instance, 65% known. Then, by leaving only the question node dynamic, we could mimic Reye's [27] approach by directly setting the probabilities for each time slice, with a controlled level of uncertainty.

6.4.4. No temporal nodes

The last option is to abandon the idea of time entirely (this does not mean we should ignore the concept of interaction history, though). One way to do this is to assign multiple question nodes to each concept (i.e. one node for each question). However, in order to preserve the idea of guesses and slips, and merge the new approach with the rest of the architecture seen so far, we would need to add another, intermediate layer (as the direction of the edges pointing to question nodes would have to be reversed). Figure 25 shows what this new substructure would look like for an answer collector with three questions.

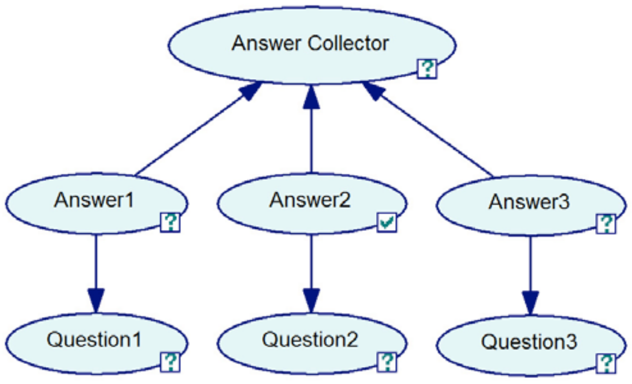


Figure 25: Multiple questions, no time slices

6.4.5. How many questions?

An obvious question that remains to be answered is how many questions we need to ask for each concept in order to mark it as "known".

- *Concept types* – First of all, we need to be aware of the number of possible question types the question generator can ask for each concept. This can also serve as a relevance measure for each concept type. For instance, QUAGS can ask only one question about a quantity space, but multiple questions about magnitudes, derivatives, (in)direct influences etc.
- *Domain difficulty* – When it comes to QR models, domain difficulty could be measured in terms of model, or even simulation complexity. However, an appropriate complexity measure would have to be a relative one, taking into consideration a large library of models, scaling from simple domains to complex ones. We are saying "simple" and "complex", instead of "small" and "large", as a large model is not necessarily a complex one, as it may produce a simpler state graph than a considerably smaller model.
- *Quantity spaces vs. network size* – As only a single question can be asked about a quantity space, not taking quantity spaces into consideration as separate concepts is also an option. Each quantity node (both generic and scenario specific) requires adding an additional node to represent its quantity space. Therefore, by excluding such nodes, we would be reducing the size of the network by a number corresponding to the total number of quantity nodes. As a substitute solution, we could attribute quantity space specific questions to magnitudes.

6.4.6. Low-level concept impact

In the simplest setup, we assume each of the low-level concept nodes contributes to the higher-level nodes equally. In the example seen in Figure 21, for instance, that would mean that each of the four nodes connected to the Size node would "weigh" 25% in terms of knowledge. However, one could disagree that the concepts are equally important, and say that, for example, the knowledge about a magnitude is more important than the one about an influence. This could be regarded as a matter of preference.

6.4.7. Answer collector impact

One might argue that an instance specific answer collector should "share" more knowledge with the instance it belongs to, i.e. have a little more impact on it than the other collector nodes. We can try to assign a weight, e.g. X, that will help us distribute the probabilities less "fairly". For example, in the above mentioned example with 3 instances, with a fair conditional probability distribution (CPD), the definition table for *PositiveProportionality AH1* looks as given in Table 1 (K = Known, U = Unknown).

PositiveProportionality_AH1								
PP AH1 Answer Collector	K				U			
PP AH2 Answer Collector	K		U		K		U	
PP AH3 Answer Collector	K	U	K	U	K	U	K	U
Known	1	0.67	0.67	0.33	0.67	0.33	0.33	0
Unknown	0.33	0.33	0.67	0.33	0.33	0.67	0.67	1

Table 1: CPD table for the PositiveProportionality AH1 node

Therefore, if the instance specific answer collector is known, the algorithm should boost the probabilities by the amount assigned to the weight variable, and if it is unknown, the probabilities should be decreased by the same amount. To be more specific, currently, the probability of a node being known (K) is calculated as follows:

$$K = k/N$$

where N is the number of instances and k the number of known instances. By altering the distribution process as explained above we get:

1. If the instance specific answer collector is known: $K = k/N + X/N$
2. If the instance specific answer collector is not known: $K = k/N - X/N$

We normalize the weight (X/N), so the probabilities get boosted/decreased by a normalized "fraction of knowledge" rather than by a fixed number (we do not want to add/subtract the same amount, e.g. 5%, to an instance with 1 and 10 duplicates). The modified CPD for X = 0.06 can be seen in Table 2.

PositiveProportionality_AH1								
PP AH1 Answer Collector	K				U			
PP AH2 Answer Collector	K		U		K		U	
PP AH3 Answer Collector	K	U	K	U	K	U	K	U
Known	1	0.73	0.73	0.39	0.61	0.27	0.27	0
Unknown	0	0.27	0.27	0.61	0.39	0.73	0.73	1

Table 2: CPD table for the PositiveProportionality AH1 node

Figure 26 depicts this idea. The displayed structure uses the non-dynamic approach (i.e. a fixed number of questions) shown in Figure 19 for the sake of clarity. We can see that one of question nodes was marked as known. Here, we can also see the guess factor at work – the reason the answer node right above the question one shows the probability of 99% instead of 100%, as one may expect, is the fact that the probability of a guess is set to 0.01 (i.e. 1%). This further boosts the probability of the instance-specific answer collector to 74% and the probability of knowing that particular instance (*PositiveProportionality AH1*) to 60%. At the same time, the other instances also receive an increase in probability, but the impact of the external answer collector is slightly less, as expected, so the new value for each of the duplicates is 57%. Each of the instances contributes equally to the generic concept, so the probability of knowing *PositiveProportionality AH* is now 58%.

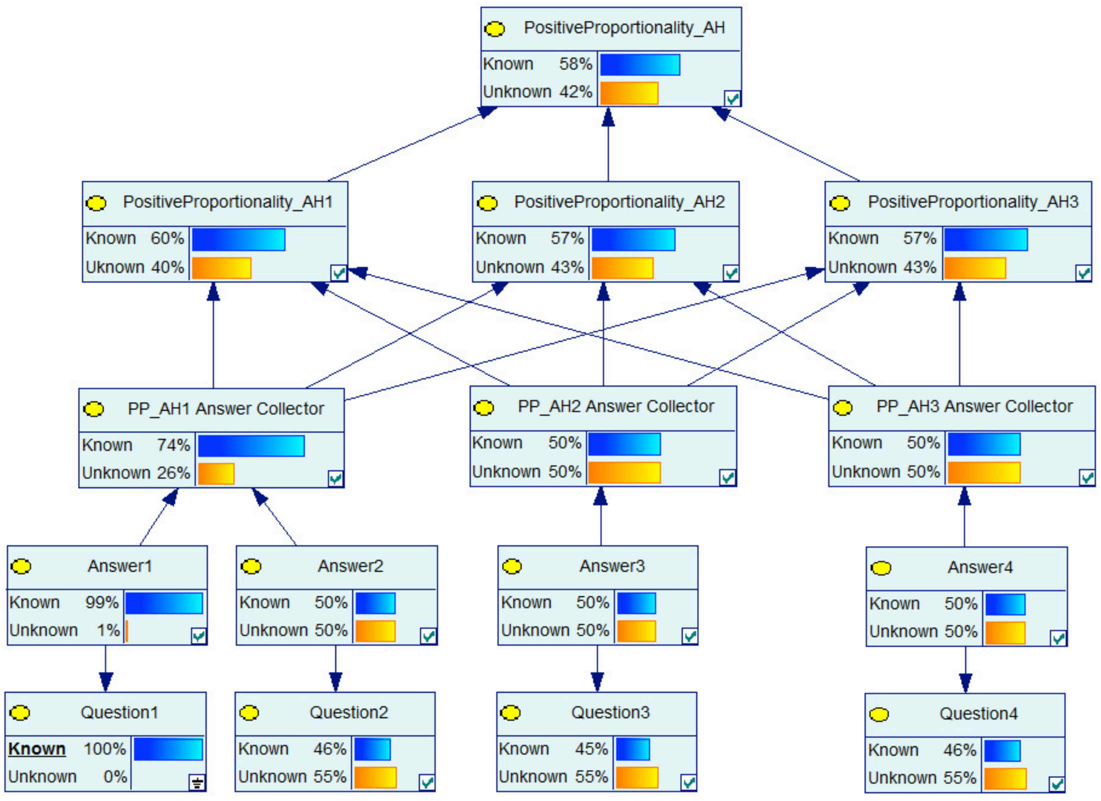


Figure 26: Answer collector impact with instance specific weights

6.5. Bayesian network structure evaluation

For the sake of evaluating our work, we have chosen the well-known Communicating vessels model [20]. It is considerably more complex than the Tree and shade model, and it contains a number of issues discussed so far which were not covered by earlier approaches, including inequalities, equations and recurring substructures.

The domain is represented by multiple containers containing a homogeneous fluid, connected to each other at the bottom by a pipe. If there is more liquid in any of the vessels, the difference in hydrostatic pressure will make the liquid flow out of the container with the higher amount of content into the other vessels via the pipe, until it eventually finds an equilibrium (balances out to the same level in all of the containers). The pressure depends only on how far the liquid surface is from the bottom of the container (the liquid is acted upon by gravity), and is not affected by the shape of the container.

6.5.1. Communicating vessels model fragments

The structure and behavior of the communicating vessel system is described by two MFs. The Contained liquid MF (Figure 27) models a single container with liquid. The quantities Amount, Height, and Pressure are introduced as consequences of a Liquid entity, contained inside a Container. Since we know that if Amount changes, Height changes in the same direction, but so does Pressure, we model these relationships using positive proportionalities in appropriate directions. Moreover, all of the quantities also have to be in the same interval at the same time. This is modeled using quantity space correspondences. Finally there is an equality relation between Height and Pressure.

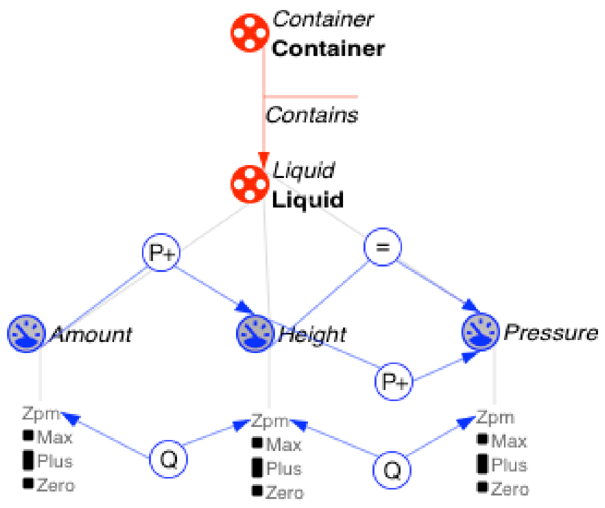


Figure 27: Contained liquid model fragment

The second MF, named Liquid flow describes the communicating vessels mechanism for two containers, and is given in Figure 28. The Contained liquid MF serves as a condition twice, i.e. once for each container. A conditional Pipe entity instance connects the two containers, and introduces the Flow quantity, which, when in the positive interval, increases Amount in the right container and decreases the one in the left one via a positive and a negative influence relationship, respectively. The actual magnitude of Flow is calculated as the difference between the magnitudes of the two Pressure quantities, i.e. by subtracting Pressure on the right side from the pressure on the left side. In order for the simulation engine to be able to determine Flow's derivative, the change is described using a positive proportionality from Pressure on the left hand side, and a negative one from Pressure on the right hand side. That is, when the pressure of the left container increases, the flow in the pipe increases as well, and when the pressure in the right vessel increases, the liquid flow decreases.

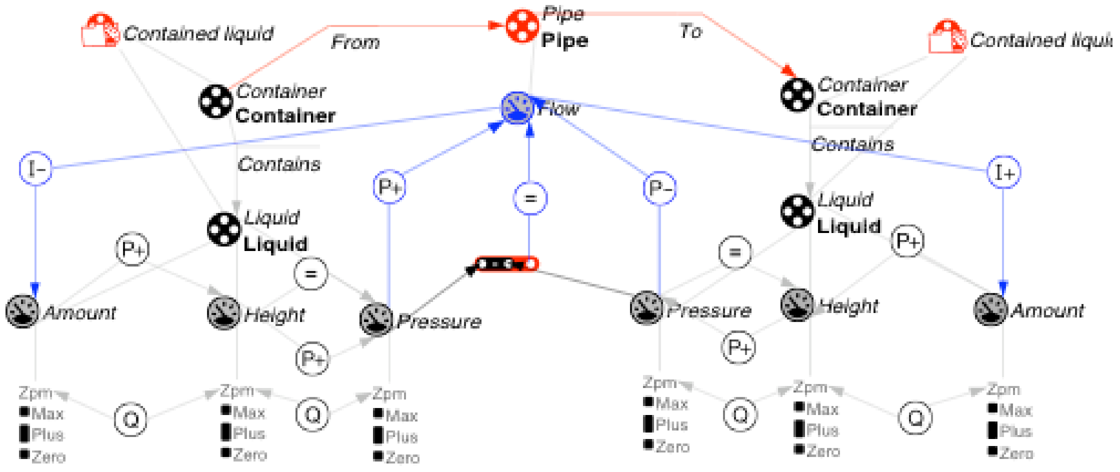


Figure 28: Liquid flow model fragment

6.5.2. Communicating vessels scenario

We will take only a single scenario into consideration and construct our network based on it. The scenario is called *'Both tanks partially filled but left is higher'* and is shown in Figure 29. It represents two containers, partially filled with oil (magnitudes set to plus), where the height of the liquid column on the left is greater than the one on the opposite side (modeled using an inequality relationship).

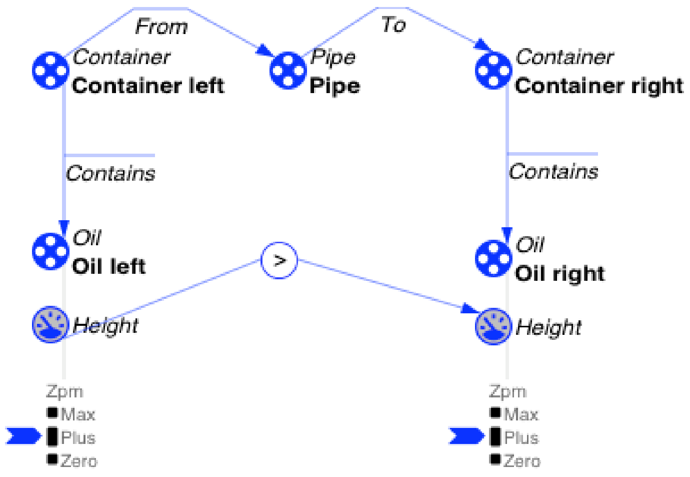


Figure 29: Both tanks partially filled but left is higher scenario

Final structure implementation: Due to space constraints and the size of the network, the final structure will not be shown here. Our choice of options was constrained due to a number of bugs, and a lack of support for certain concepts (at the time of this writing) in the jSMILE API.

To be more specific, the concepts that could not be implemented were the ideas of temporal nodes and soft evidence. Therefore, we were left with only one option, namely, to abandon the concept of time and determine the number of possible question nodes beforehand. Despite these shortcomings, however, the approach yielded promising results (see section 6.5.4) and once the problems in the jSMILE API are addressed, we plan to implement the missing concepts.

6.5.3. Choice of parameters

Table 3 lists the choice of parameters used for evaluation of the network. The prior probabilities of all concept nodes are set to 50%, i.e. the probabilities of a student knowing or not knowing a topic are the same. Furthermore, we assume that all parent (root) nodes of a concept contribute equally to the child (concept impact entry). For instance, if an entity has four quantities, each quantity node will be "worth" 1/4, i.e. 25% in terms of knowledge. The probability of a learner giving a correct answer, although he does not know it (i.e. guessing) is 0.1. The probability of answering incorrectly, while knowing the topic (i.e. slipping) is 0.01. Finally, the last seven entries in the table represent an experimental set of choices for the number of questions per concept, based on the number of possible questions that can be asked regarding each of those concepts by QUAGS.

Parameter	Parameter value
Concept node prior	0.5
Concept impact	1/#parents
Answer collector weight	0.06
Probability of a guess	0.1
Probability of a slip	0.01
Questions per magnitude	5
Questions per derivative	5
Questions per influence	5
Questions per proportionality	5
Questions per correspondence	2
Questions per inequality	2

Questions per calculation	1

Table 3: Choice of parameters

6.5.4. Evaluation

We first evaluate a single root node to see how its probability responds to evidence updates, and how this knowledge is propagated to upper level nodes. For each concept tested, we first visit all of the question nodes once, and arbitrarily add evidence. Then, we revisit the same nodes, and update any answer that was initially set to be incorrect, to correct, so the probability of the root node reaches its maximum value (for its answer collector; duplicate instance nodes are affected by other duplicates' answer collectors, too). Figure 30 shows how the system's belief of the user's knowledge of *Magnitude_s_7*[1] and *owl_q_s_flow1* (the quantity the magnitude belongs to) develops across eight learner-system interactions/questions. As given in Table 3, since the node is of type *Magnitude*, we ask five questions. The learner gives an incorrect answer to the third question, and the probability of both the magnitude and quantity concepts decreases. The remaining two answers are correct, though, and the probabilities go up. Then, as described above, we "fix" the single incorrect answer at interaction number six, so the concept of *Magnitude_s_7* becomes 91%, which is the maximum value that node can attain, due to the guess and slip factors and the fixed number of questions. The *owl_q_s_flow1* quantity node reaches 56%, which is the maximum probability for that node if there is no evidence about the other parents of the same node.

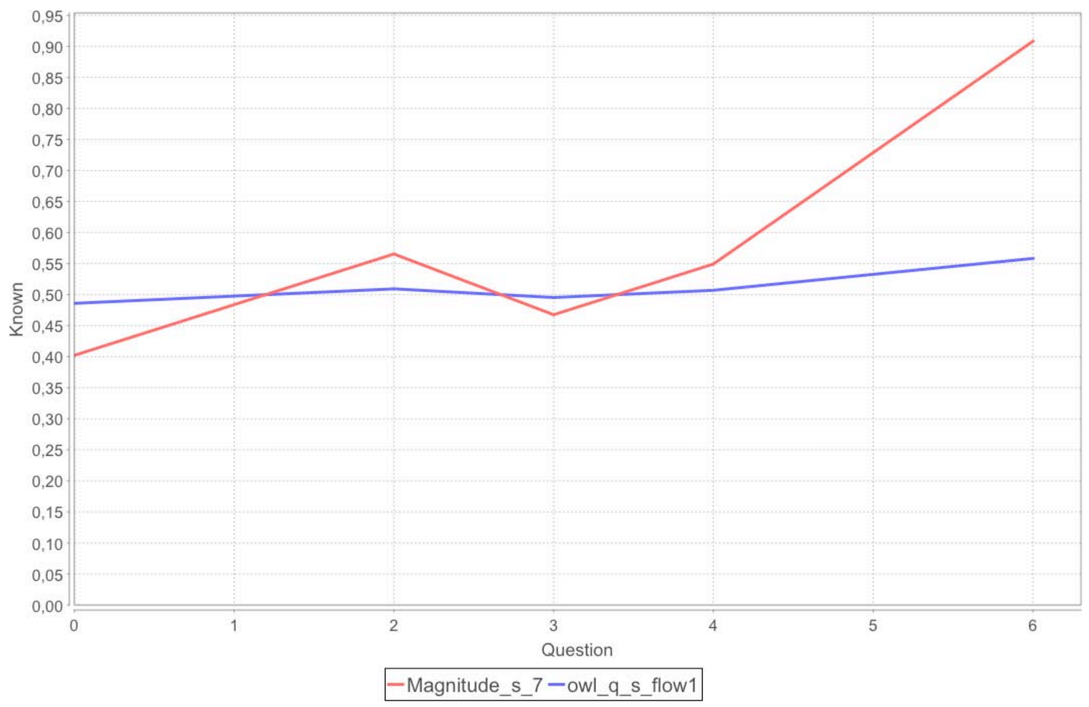


Figure 30: Learning chart for the flow quantity's magnitude

The above test shows how the system's belief progresses when we are dealing with a single instance with no duplicates, i.e. only one answer collector. Next, we observe the behaviour of a quantity instance, which is not unique, namely, *owl_q_s_amount1* through its magnitude, i.e. *Magnitude_s_3* (Figure 31).

[1] The evaluation graph names correspond to the internal representation of each concept in OWL, as the network construction process has been automated. The "_s" suffix indicates we're dealing with a scenario specific concept, i.e. an instance.

Because the first four answers are all wrong, the probability of knowing the magnitude goes down to only 24%. Meanwhile, the probability of knowing the quantity it belongs to drops down to 45%, and the second instance of the *Amount* quantity decreases slightly less, i.e. to 46%, due to the answer collector impact parameter. Then, the remaining question is answered correctly, and the examination session is extended by another four interactions, so all of the answers are fixed. The magnitude concept gets to 72% after interaction number nine, which is the maximum it can reach, as the second instance holds the remaining knowledge. At the same time, *owl_q_s_amount1* increases to 54%, and the second instance to around 53%, which is now slightly less, but again, due to the answer collector impact.

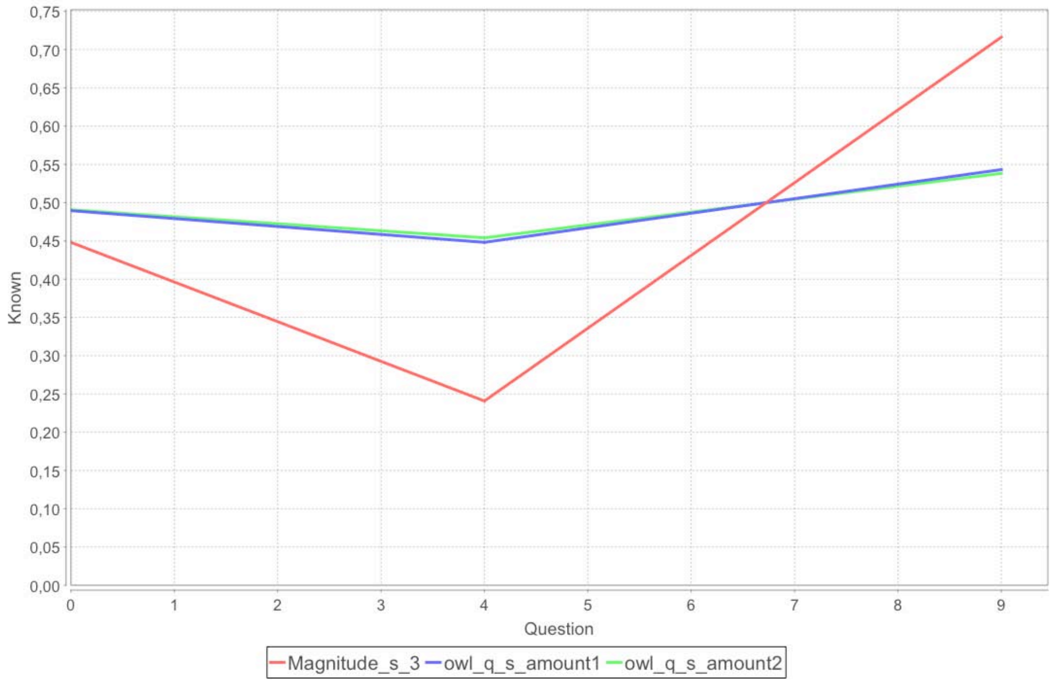


Figure 31: Learning plot for the amount quantity's magnitude

Instead of a root node, we now turn the focus to a whole quantity, namely, *owl_q_s_flow1*. We test all concepts related to the quantity using the same method as seen above, and watch the system's belief about both the parent nodes and the child develop. We also see the system asks a single question about *Minus_s_1*, as the quantity is part of the equation seen in Figure 32.The probability of knowing the quantity slowly progresses towards the maximum (91%).

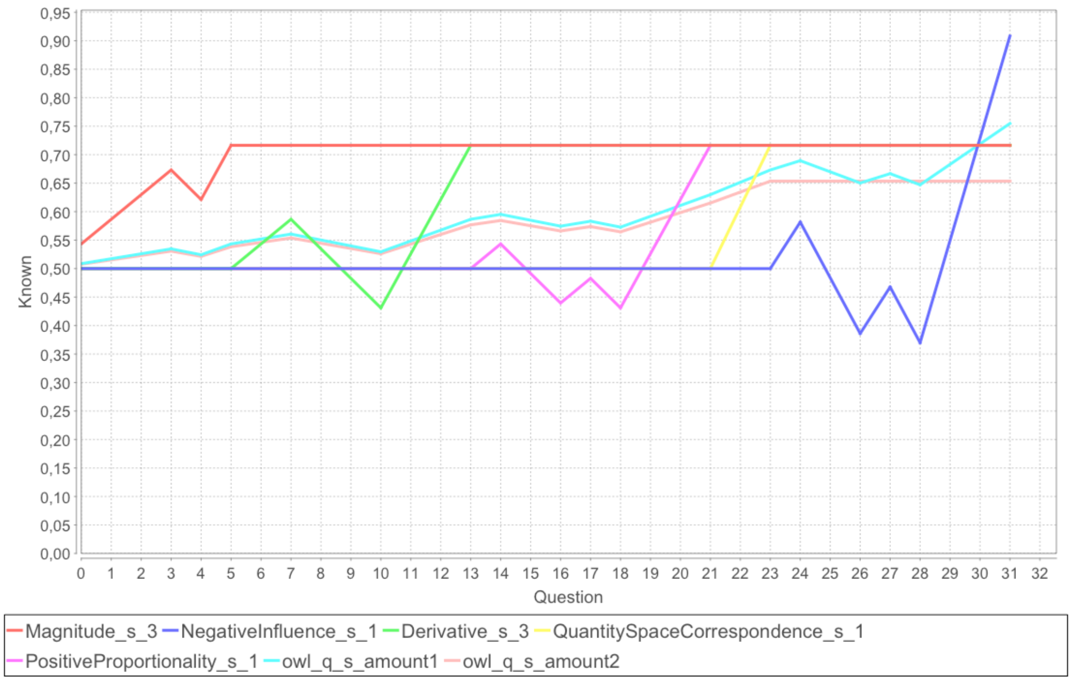


Figure 32: Learning plot for the amount quantity and related concepts

In Figure 32 we show the results of a session focusing on the quantity *owl_q_s_amount1*. We test all quantity related concepts, and show the results for both the quantity and the second instance of the *Amount* quantity (*owl_q_s_amount1*). We see that as the probability of knowing first instance increases, so does the probability of understanding the second one, only at a slightly slower rate. The breaking point is at question number 24, where the system's belief about the second instance stops changing. This is due to the fact that questions 24–28 are related to the concept of *NegativeInfluence_s_1*, i.e. the negative influence relationship between the quantities *Flow* and *Amount* (Figure 28). This information is only *Liquid flow* MF specific and, as such, carries no information about the second instance of the *Amount* quantity.

The small evaluation presented here shows the potential and workings of the approach taken. We expect that use in actual educational settings will further refine this picture.

7. Conclusion

In this deliverable we presented the progress and current state of task 5.2. “Design and implementation of dialog functionality for characters”. We gave an overview of the state of art in dialog management in learning environments and described the three different components of our approach: Dialog Management, Verbalization and User Modeling.

This dialog functionality can now be used for the implementation of the virtual characters’ dialogs for future use cases. It allows learners to have interesting and engaging dialogs with the characters. The characters can react to the learners’ actions (past and current) and refer to conceptual knowledge in natural language.

But in addition, owing to SceneMaker’s part in this approach, teachers and other interested parties will be able to change, modify or enhance these dialogs in a comfortable way.

8. Discussion and Future Work

In this final chapter, we reflect on each of the parts of the characters' dialog functionality presented in this deliverable.

8.1. Dialog Management

The choice of SceneMaker as the dialog manager for our characters allows us to create their behavior in an intuitive way while also taking the learners' actions into account. It also comes with a couple of secondary benefits such as the possibility for multi-language dialogs, separation of logic and content and the opportunity for non-computer experts to create new dialogs in an easy way.

Through the implementation of the Teachable Agent use case (see Deliverable 5.3) we learned that our approach is viable for combining the actions of multiple characters with the learner's interaction and conceptual knowledge from the CM.

Our mockup implementation of the Diagnosis use case showed how the different teaching strategies can be used to guide a learner in the right direction and provide valuable help. This feature will become interesting in the upcoming Deliverable 5.4, when we will use it to implement the "real" Diagnosis and the Ontology-Based Feedback.

Finally, it is left to say that future use cases might bring new challenges that require some alterations to the usage of the SceneMaker within DynaLearn, new commands and tags being the most probable ones.

8.2. Verbalization

While the verbalization process described in this report suits the needs of the dialog functionality, it is a very simple one. So far, it was used and showed good results for verbalizing conceptual knowledge for the Basic Help and Teachable Agent use cases.

Future use cases might warrant improvements and the addition of more sophisticated concepts. However, as supporting other languages (Brazilian being the most prominent one) is one of the future aims, it has to be seen, which of these concepts can be used for multiple different languages.

8.3. User Model

A set of fundamental issues related to constructing a BN from a QR model were successfully solved. The new approach advances this work. It provides the system with the means to successfully gather information about a learner's understanding of a domain. Still, it appears some obstacles need more work. For instance, selecting an appropriate number of questions per concept type beforehand needs to be further investigated. The lack of temporal nodes makes it impossible to see the learner's knowledge of any concept go above a limit set/constrained by the guess and slip parameters.

Further improvement concerns alternative methods for updating a user's knowledge (both temporal nodes and soft evidence). This would solve the above-mentioned problems regarding the maximum

probability that can be achieved for any node. Also, the idea of different concept types carrying different amounts of information should be given more attention.

Another topic that should be investigated further is the idea of "landmarks". For instance, certain MFs have conditions, such as a specific quantity value that needs to be reached in order for the MF to get activated. This information appears to be crucial, and, as such, should be properly represented in the learner model.

References

[1] Cade, W.L., Copeland, J.L., Person, N.K., D'Mello, S.K.: Dialogue Modes in Expert Tutoring. In: Proc. of the 9th Int. Conference on Intelligent Tutoring Systems, pp 470-479, 2008

[2] Lipscomb, L., Swanson, J., West, A.: Scaffolding. In: Emerging Perspectives on Learning, Teaching and Technology, 2001

[3] Vygotsky, L.: Mind in Society. Havard University Press, 1978

[4] Zhao, R., Orey, M.: The scaffolding process: Concepts, features, and empirical studies. University of Georgia, 1999

[5] Graesser, A.C., Person, N.K., Harter, D. and The Tutoring Research Group: Teaching Tactics and Dialog in Autotutor. In: Artificial Intelligence in Education (12), pp 257-279, 2001

[6] Allen, J. F., Byron, D. K., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A.: Towards conversational human-computer interaction. In: AI Magazine**, 22(4), pp 27-37, Winter 2001**

[7] Gebhard, P., Kipp, M., Klesen, M., Rist, T.: Authoring scenes for adaptive, interactive performances. In: Proc. of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, pp 725-732, 2003

[8] Harel, D.: Statecharts: A visual formalism for complex systems. In: Science of Computer Programming, volume 8, pp 231-274, June 1987

[9] Spierling, U., Iurgel, I.: Just talking about art - creating virtual storytelling experiences in mixed reality. In: Proc. of the Second International Conference on Virtual Storytelling, pp 179-188, 2004

[10] Mehlmann, G. U.: Scenemaker 3: An interpreter for parallel processes modeling behavior of interactive virtual characters. Master's thesis, Saarland University and German Research Center for Artificial Intelligence, 2009

[11] Müller, W., Spierling, U., Weiß, S.: Synchronizing natural language conversation between users and multiple agents in interactive storytelling applications. In Proc. of TESI 2005 Conference, Kent, 2005

[12] Spierling, U.: Interactive digital storytelling: Towards a hybrid conceptual approach. In: Proc. of Digital Games Research Association’s 2nd International Conference: Changing Views - Worlds in Play, 2005

[13] Galanis, D. and Androutsopoulos, I.: Generating Multilingual Descriptions from Linguistically Annotated OWL Ontologies: The NaturalOWL System. 11th European Workshop on Natural Language Generation, pages 143–146, 2007

[14] Gatt, A. and Reiter, E.: SimpleNLG: A realisation engine for practical applications. 12th European Workshop on Natural Language Generation, pp 90-93, 2009

[15] Wilcock, G.: An Overview of Shallow XML-based Natural Language Generation. In: Proc. Of the Second Baltic Conference on Human Language Technologies, pp 67-78, 2005

[16] Horridge, M., Bechhofer, S.: The OWL API: A Java API for Working with OWL 2 Ontologies. OWLED 2009, 6th OWL Experienced and Directions Workshop, 2009

[17] Baker, R. S., Corbett, A. T., Aleven, V.: More accurate student modeling through contextual estimation of the node slip and guess probabilities in Bayesian knowledge tracing. In: Proc. of ITS 2008, pp 406–415, 2008

[18] Beck, J. E., Chang, K.-M.: Identifiability: A fundamental problem of student modeling. In: Proc. of UM 2007, pp137–146, 2007

[19] Beck, J. E.: Difficulties in inferring student knowledge from observations (and why you should care). In: Educational Data Mining: Supplementary proc. of AIED 2007, pp 21–30, 2007

[20] Bredeweg, B., Bouwer, A., Liem, J., Salles, P.: Curriculum for learning about QR modelling. Naturnet-Redime, STREP EU FP6 project no. 004074, D6.9.1., 2006

[21] Bredeweg, B., Linnebank, F., Bouwer, A., Liem, J.: Garp3 - workbench for qualitative modelling and simulation. Ecological Informatics 4(5–6), pp 263–281, 2009

[22] Brielmann, M.: A learner model based on a Bayesian network for Garp3. Master's thesis, AMSTEL Institute, University of Amsterdam, 2009

[23] Corbett, A. T., Anderson, J. R.: Knowledge tracing: Modelling the acquisition of procedural knowledge. User Modeling and User-Adapted Interaction 4(4), pp 253– 278, 1995

[24] Goddijn, F., Bouwer, A., Bredeweg, B.: Automatically generating tutoring questions for qualitative simulations. In: Proc. of QR 2003, pp 87–94, 2003

[25] Koedinger, K. R.: Toward evidence for instructional design principles: Examples from cognitive tutor math 6. invited paper. In: Proc. of PME-NA XXXIII 2002, pp 21–49, 2002

[26] Millán, E., Pérez-de-la Cruz, J.-L., Suárez, E.: Adaptive Bayesian networks for multilevel student modelling. In: Proc. of ITS 2000, pp 534–543, 2000

[27] Reye, J.: Student modelling based on belief networks. International Journal of Artificial Intelligence in Education 14(1), pp 63–96, 2004

[28] Valtorta, M., Gyun Kim, Y., Vomlel, J.: Soft evidential update for probabilistic multiagent systems. International Journal of Approximate Reasoning 29, pp 71–106, 2000