Deliverable number:	D5.3
Deliverable title:	Basic Help and Teachable Agent

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, René Bühling, Wouter Beek, Floris Linnebank, Jochem Liem, 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

In this document we present two different use cases of the DynaLearn software which involve virtual characters: The Teachable Agent and the Basic Help.

We elaborate on how the necessary knowledge for these use cases is extracted from the CM component, how it is represented and how it is turned into presentations by the virtual characters that are engaging and helpful for the learners. We also describe how the different parts of the dialog system presented in Deliverable 5.2 aid us in achieving this task.

Internal reviewers

Paulo Salles (FUB), Institute of Biological Sciences, University of Brasilia

Esther Lozano (UPM), Ontology Engineering Group, Universidad Politécnica de Madrid

Acknowledgements

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 draft	2010-07-14	Wißner,Häring,Bühling
02	Input by UvA	2010-07-16	Beek,Linnebank,Liem,Bredeweg
03	Draft for review by partners	2010-07-19	Wißner, André
04	Integrated suggestions from review by FUB	2010-07-26	Wißner
05	Integrated suggestions from review by UPM	2010-08-17	Wißner,Beek
06	Final version	2010-08-20	Wißner,Bühling

Contents

Abstract 2

Internal reviewers 2

Acknowledgements 2

Document History 3

Contents 4

1. Introduction 6

2. Basic Help 7

2.1. Introduction and different Types of Help 7

2.2. Interaction Flow and Examples 7

2.3. Knowledge Extraction and Representation 9

2.3.1. Requirements for Basic Help knowledge representation and extraction 9

2.3.2. General aspects of the Basic Help knowledge representation and extraction 10

2.3.2.1. Modularity 10

2.3.2.2. Knowledge linking 10

2.3.2.3. Natural language 11

2.3.3. Specific aspects of the Basic Help knowledge representation and extraction 11

2.3.3.1. "What is?" 11

2.3.3.2. "How to?" 12

2.3.3.3. "Why?" 15

2.4. Dialog Management 16

2.4.1. Sceneflow and Example Scenes 16

2.4.2. Usage of Verbalization Module 17

2.4.3. Usage of User Model 18

3. Teachable Agent 19

3.1. Introduction and Purpose 19

3.2. Learning by Teaching with a Teachable Agent 19

3.3. Interaction Flow and Examples 20

3.4. Knowledge Extraction and Representation 27

3.5. Dialog Management 27

3.5.1. Sceneflows and Example Scenes 27

3.5.2. Usage of Verbalization Module 30

3.5.3. Usage of User Model 30

4. Conclusion 31

5. Discussion and Future Work 32

References 33

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 second item, namely the results of Task 5.3 "Basic help and teachable agent".

The remainder of this document is structured as follows: The next two chapters will discuss in details both the Basic Help (2) and the Teachable Agent (3). We will end with a conclusion (Chapter 4) as well as a discussion and an outlook of future work (Chapter 5).

2. Basic Help

2.1. Introduction and different Types of Help

The Basic Help support functionality communicates the knowledge pertaining to those aspects of the DynaLearn interactive learning environment (ILE) that are visible to the user and that the user can directly interact with. For any one of these visible aspects, the knowledge communicated explains the direct properties as well as the embedding of the visible aspect within the broader context of the rest of the ILE. There are three kinds of directly visible aspects, and they are distributed over the three kinds of Basic Help:

- **"What is?" handles the knowledge regarding the diagrammatic representation of the user-created model. The visible aspects are the various modeling ingredients that constitute the on-screen structure of the model.**
- **"How to?" handles the knowledge regarding the screens, dialogs and buttons that the user interacts with. The visible aspects can either be the screens themselves, or modeling ingredients.**
- **"Why?" handles the knowledge regarding the diagrammatic representation of the simulation results. The visible aspects are the visual representation of magnitude and derivative values, as well as states and state transitions that are visualized in the state graph simulation environment.**

Even though individual Basic Help requests focus on relatively isolated pieces of model and workbench knowledge, these model ingredients and workbench components are always inherently linked to the larger fabric of the complete model and the overall workbench. The strategy of the Basic Help functionality is therefore to be concise and focused with respect to individual knowledge requests, but to allow for more complexity by adding hyperlinks that can be optionally clicked in order to gain a broader understanding of the material.

2.2. Interaction Flow and Examples

For this example, let us assume that the learner has opened the Communicating Vessels model, as shown in Figure 1.

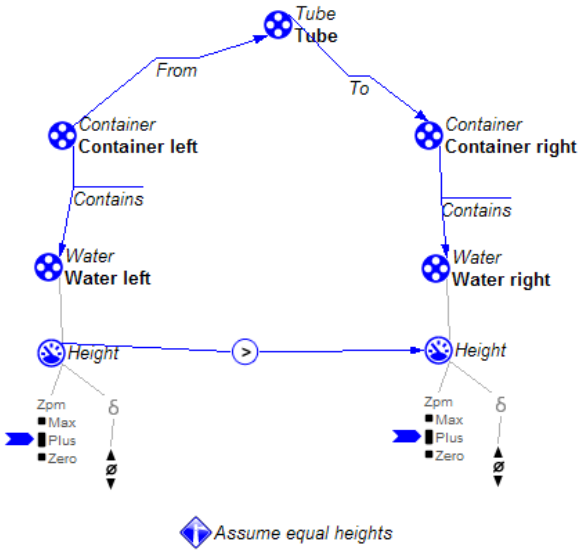

Figure 1: Communicating Vessels model

Now he asks "What is Water right?" by selecting the appropriate item from the menu. The Teacher character will then appear and, after a short introduction, offer the desired explanation (see Figure 2).

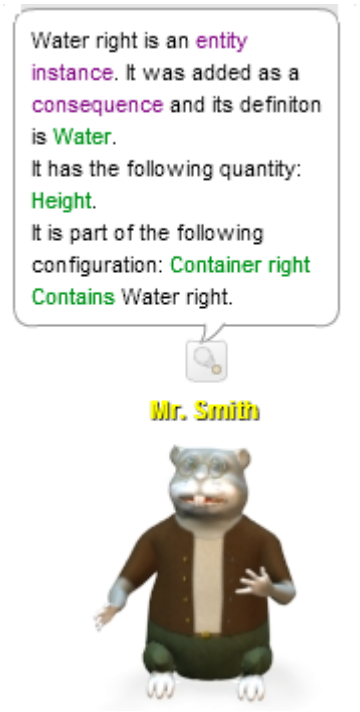

Figure 2: The Teacher character giving an explanation about an entity instance

If learners want further information, they have two possibilities of asking follow-up questions: One is to select a purple term from the Teacher's answer, which will bring up the glossary, explaining the term. This explanation can in turn contain glossary terms. See Figure 3 for an example of the glossary.

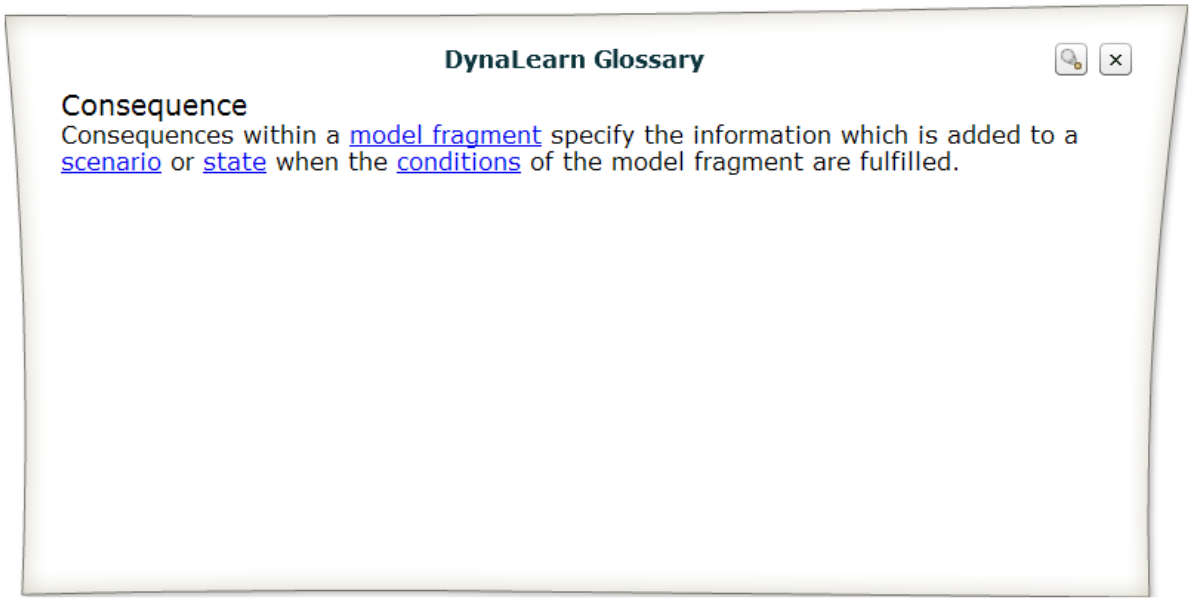

Figure 3: Glossary, describing the term "Consequence"

The other possibility for a follow-up question is to click on a green term. This will generate a new What is? question about this term and the Teacher will again give the explanation.

The interaction flow for "How to?" and "Why" questions is similar to the one for the "What is?" question shown here: The learner selects something (a model ingredient, a state in a simulation etc.) and the possible questions will be listed in the support menu. Upon choosing one of them, the character will appear and give the answer, as well as possibilities for follow-up questions of the same kind.

2.3. Knowledge Extraction and Representation

This section discusses the knowledge representation and extraction of the Basic Help component. In section 2.3.1 we formulate the requirements which the knowledge representation and extraction should adhere to. Section 2.3.2 describes how these requirements have been met by the Basic Help design. Section 2.3.3 focuses on those aspects of the knowledge representation and extraction that are peculiar to the three Basic Help variants, i.e. the "What is?", the "How to?", and the "Why?" functionality.

2.3.1. Requirements for Basic Help knowledge representation and extraction

Since support information should be made available immediately based upon the learner's interaction, complex information extraction and/or logical derivation methods should be avoided as much as possible. Ideally, the representation of the support knowledge should make complex processing superfluous by being particularly tailored towards the support effort.

Because the Basic Help knowledge should be communicated to the user in natural language, the knowledge architecture of the support functionality should provide the ingredients that allow the generation of a natural language dialogue.

This gives us the following four requirements to which the Basic Help knowledge representation and extraction architecture should adhere:

- **The transmitted knowledge should always be both concise and focused on one particular aspect of a model or the workbench.**
- **The knowledge should be immediately available to the user.**
- **The user should have the ability to access broader knowledge as an option.**
- **The knowledge must be communicated in natural language.**

2.3.2. General aspects of the Basic Help knowledge representation and extraction

This section discusses the way in which the requirements of the previous section have been met by the general Basic Help design.

2.3.2.1. Modularity

The first two points can be covered both at the same time, since a knowledge representation in which information is formulated in a concise and focused manner, does not require much additional information retrieval or logical derivation. The reasoning is largely being incorporated into the representation. In order to achieve this, a modular representational approach is used.

In order to be modular, each ingredient is associated with its own RDF/XML structure. This means that the granularity at which the Basic Help information requests are processed, i.e. individual model ingredients, is the same level at which the model has been represented in the Basic Help knowledge representation. This makes the representation in close agreement with the extraction process, in the sense that no additional information retrieval or other complex methods are required.

Normally, knowledge in the DynaLearn software is represented using OWL. However, we nevertheless chose to use this special RDF/XML structure for the following reasons: The logical restrictions that the OWL language poses are too stringent for the intended purpose. Also, the existing OWL representation was not modular. This is partly due to the many interconnections that the OWL language possesses, making the representation more convoluted than it should be for the intended purpose.

2.3.2.2. Knowledge linking

Since a support interaction between a learner and a teacher typically consists of a to-and-fro between learner requests and teacher elucidations, we want to address the third requirement, as identified in the previous section, by incorporating the ability to link to further Basic Help requests from within the explanation of the previous Basic Help elucidation.

These knowledge links are represented as RDF linked data URIs. In the knowledge extraction process they are directly mapped onto hyperlinks that figure in the output dialog which the Virtual Character communicates. Since the knowledge representation contains a modular description of every readily available visual aspect, this means that the learner can click on any of the concepts that are mentioned in the dialogue, triggering a follow-up Basic Help request.

In addition to linking through to other modular ingredients, hyperlinks are also added for terms that are specific to the DynaLearn ILE as well as to the Qualitative Reasoning methodology that the ILE is based on. These terminological hyperlinks, which are also represented as RDF linked data, are clickable in the Virtual Character's communication balloons as well.

2.3.2.3. Natural language

Another requirement (as identified above) is that the support knowledge representation allows the generation of a natural language dialogue.[1]

In order to stay close to natural language, we want to represent the knowledge in a flat, i.e. non-nested manner. In addition to avoiding nesting as much as possible, the structure of the representation should itself be similar to natural language's grammatical structure. The general form of a sentence is Subject-Predicate-Object, so that the Object term is predicated of the Subject term. The RDF/XML representation has exactly this structure.[2]

In order to make the DynaLearn ILE even more versatile with respect to language interactions, the user can choose different languages in which to encode his model. In addition to that, we allow the Basic Help component to support multiple languages as well. As of yet, only the English language (with a distinction between British and American English) is fully supported, but additional languages can be added simply by extending the number of tags in the various Basic Help components. An example of how this works is given in the "How to?" section. Tags for additional should translate the original English information and are marked with the `xml_lang` argument. All languages from the ISO 639 standard[3] **are supported.**

2.3.3. Specific aspects of the Basic Help knowledge representation and extraction

In this section those aspects of the Basic Help representation and extraction are described that are peculiar to the three different forms of Basic Help, i.e. the "What is?", the "How to?", and the "Why?" functionality.

2.3.3.1. "What is?"

The "What is?" knowledge is generated for a specific model ingredient that the user selected prior to issuing the "What is?" question. Since the model ingredients are different for every model that a user creates, this information is generated on the fly. Within a single model, the various model ingredients are identified by unique URIs. These URIs also provide the possibility to link through to additional "What is?" knowledge requests (as mentioned above).

The code example in Figure 4 shows how the representation is centered around an individual model ingredient (in this case an entity instance called 'Stove'). The represented material consists of (a) all the direct properties of this model ingredient as well as (b) all other model ingredients to which it is directly related.

[1] Artificial languages are less natural and must first be learnt, whereas the support functionality should also help starters, i.e. learners without prior knowledge of the DynaLearn ILE. Schematic representations, although potentially powerful, would not be adequate for the support functionality, since its purpose is to explain aspects of the schematic representation of qualitative systems knowledge. Introducing yet another schematic representation would thus be counter-productive. Also, natural language adds another medium to the text-based on-screen representations, thereby providing an extra channel to effectively communicate knowledge.

[2] For a more in-depth discussion of the technical details of the natural language properties of the RDF/XML representation, see deliverable 3.2, section 9.3.

[3] See the ISO 639 standard at http://www.loc.gov/standards/iso639-2/php/code_list.php

```
<rdf:Description rdf:about="entityInstance#Stove_000000068">
    <qr:hasName xml:lang="en" rdf:datatype="&xsd;#string">Stove</qr:hasName>
    <qr:hasRemarks xml:lang="en" rdf:datatype="&xsd;#string"></qr:hasRemarks>
    <qr:hasCategory xml:lang="en" rdf:datatype="&xsd;#string">entityInstance</qr:hasCategory>
    <qr:hasState xml:lang="en" rdf:datatype="&xsd;#string">consequence</qr:hasState>
    <qr:inAggregate rdf:resource="expressionFragment#Expression_fragment_000000019" name="Expression fragment"/>
    <qr:hasDefinition rdf:resource="entityDefinition#Stove_000000067" name="Stove"/>
    <qr:hasQuantityInstance rdf:resource="quantityInstance#Temperature_000000093" name="Temperature"/>
    <qr:hasQuantityInstance rdf:resource="quantityInstance#Heating_000000076" name="Heating"/>
    <qr:hasConfiguration rdf:resource="configurationInstance#Heats_000000074" name="Heats" otherName="Pan"
      otherID="entityInstance#Pan_000000070" direction="fromThisToOther"/>
</rdf:Description>
```

Figure 4: Example of "What is?" representation

Examples of direct properties are an ingredient's name, remarks, category, and state. Examples of direct relationship are having quantities, occurring in configurations, and being the instance of a definition.

For some directly linked ingredients, it is interesting to show more information than just the directly related ingredient by itself. For instance, when an entity instance is related to another entity instance via a configuration, we want to know the directedness of this relation as well. Also, we want to know the entity instance that resides on the other side of the configuration. Because of the reification of configurations, it would otherwise not be possible to refer to the other entity instance directly.[4]

2.3.3.2. "How to?"

The "How to?" functionality has two modes: with and without a model ingredient selection. In the latter case, the actions that can be performed in the current screen of the DynaLearn ILE are accessible through a layered menu. The layered menu including action categories like "Add", "Remove", and "File". Under these categories reside "How to?" requests such as "How to add an entity?", "How to remove an imported model fragment?", and "How to create a new model?".

In the second "How to?" mode, i.e. the one with model ingredients selected, the selected model ingredients work as a filter on the layered menu of "How to?" requests, showing only those that pertain to the current selection.

The "How to?" knowledge representation consists of two interlinked hierarchies. One consists of the *actions* that a user can perform in the DynaLearn ILE (Figure 5). These comprise actions the user can perform in order to alter the current model, like adding, deleting, and changing model ingredients. Other actions allow the user to open, save or print a model, simulate a model, or change a model's metadata. Lastly, there are actions that allow the user to engage with the virtual character use cases that the DynaLearn ILE offers.

[4] For a complete overview of the various "What is?" properties that are generated, see deliverable 3.2.

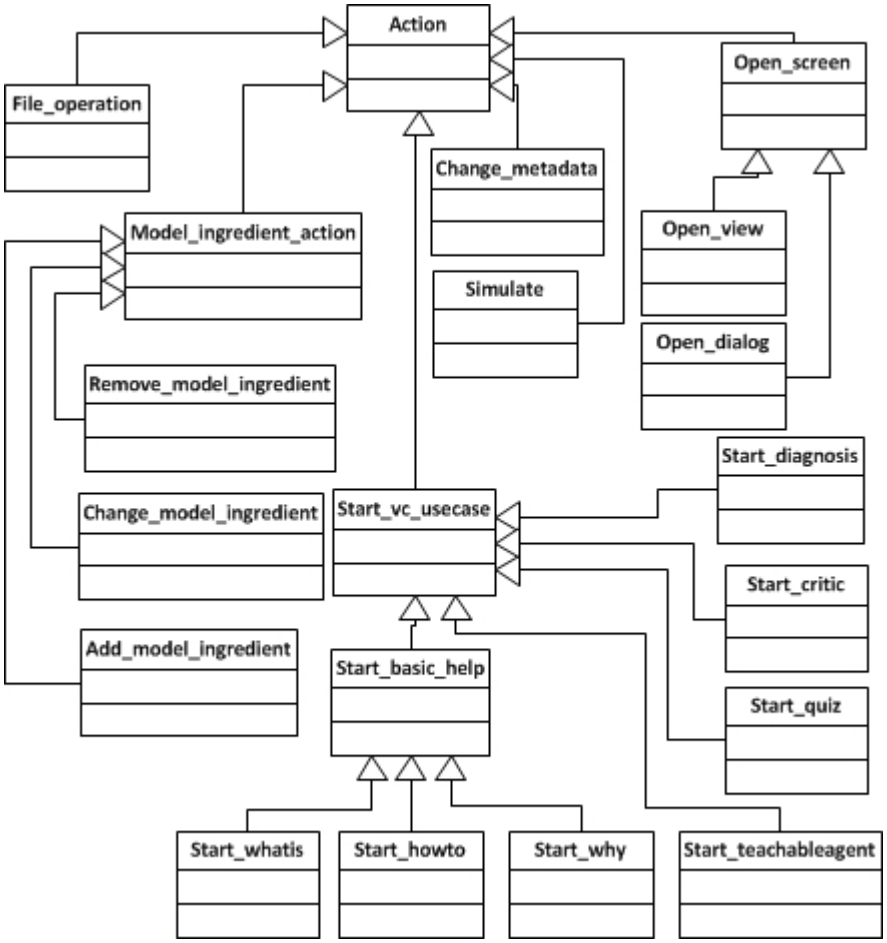

Figure 5: The action hierarchy for the "How to?" functionality

In addition to the action hierarchy, there is a hierarchy of *screens* (Figure 6). These consist of the main window, the views that can be opened inside the main window, and the various dialogs that can be opened from either the main window or from a specific view that is opened inside the main window.

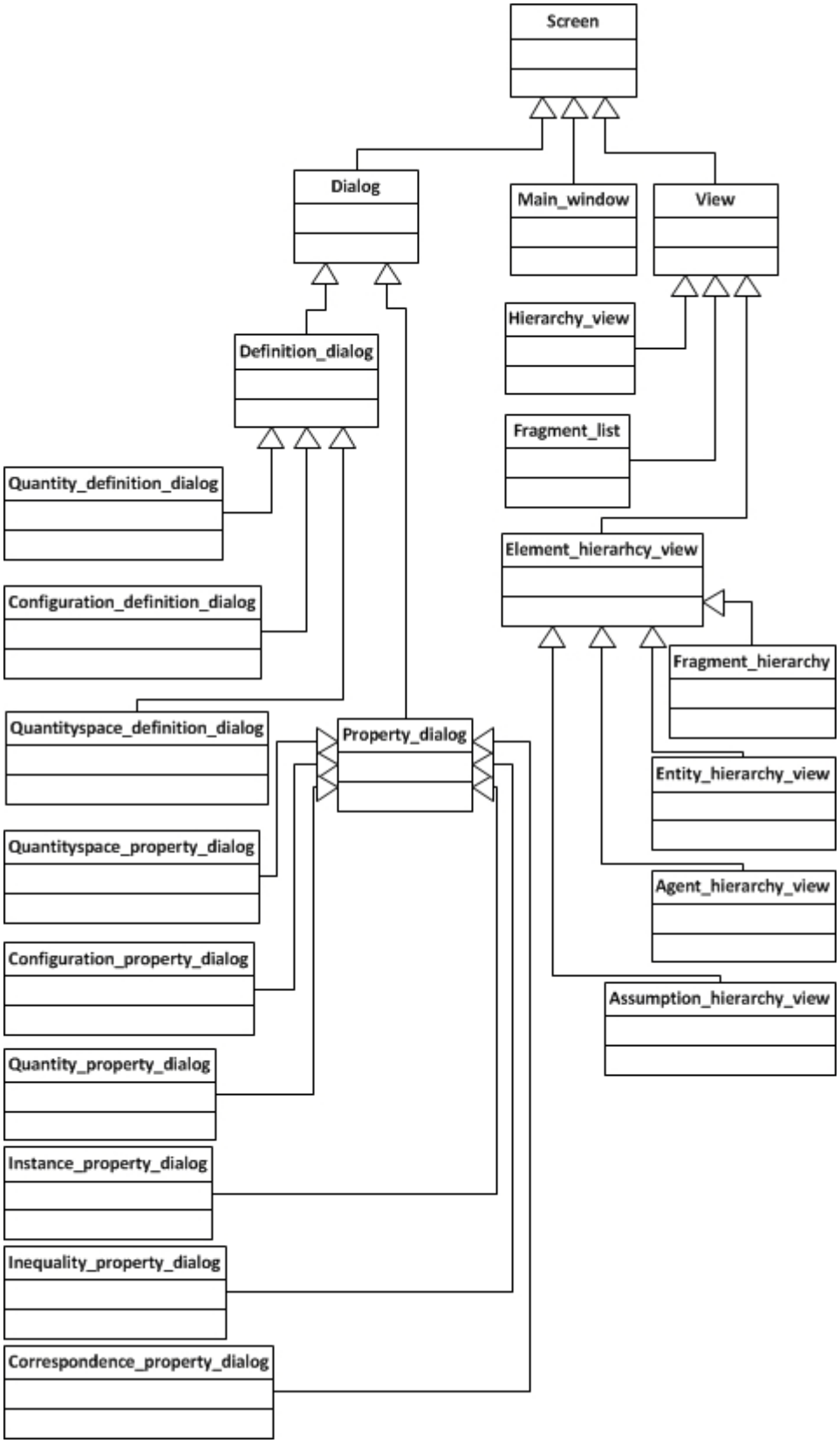

Figure 6: The screen hierarchy for the "How to?" functionality

The hierarchic structure of the actions and screen is represented using SKOS (http://www.w3.org/TR/skos-primer/), e.g. the `skos:broaderTransitive` **line in the below code snippet.**

In the knowledge representation, the screens and actions are related to each other. Every screen allows certain actions to be performed. And each action needs to be performed in an appropriate screen. For representing these interrelations we use the Basic Help namespace, called 'bh'. The tags `bh:locatedIn` and `bh:accessibleFrom` implement these interconnections.

An important aspect of the DynaLearn ILE is the different learning spaces in which the conceptual modeling experience is segmented. Because each learning space makes different functionality available, the actions that are available may differ across learning spaces. In addition to that, even if the same action is available in multiple learning spaces, then still the preconditions for performing that action, as well as the consequences that performing the action may have, can differ across learning spaces. See deliverable 3.1 "Multi Use Level Workbench" for an overview of the various learning spaces.

In order to incorporate how the functionality of the ILE is distributed over the various learning spaces, an argument is included that marks the learning spaces to which the stored knowledge applies. In the below example, three different description of the same action (i.e. adding a configuration instance) are provided. Each description describes the action in the context of different learning spaces.

As was discussed above, the DynaLearn ILE allows for multi-language support. By adding tags that contain the same information as their English counterparts in translation, it is possible to incrementally add new languages to the Basic Help functionality. In the example in Figure 7, the name of the described action has also been added in German.

```
<rdf:Description rdf:about="add_configuration_action">
        <bh:hasName xml:lang="en" rdf:datatype="&xsd;#string">Add Configuration</bh:hasName>
        <bh:hasName xml:lang="de" rdf:datatype="&xsd;#string">Konfiguration zufügen</bh:hasName>
        <bh:Description xml:lang="en" ls="1">Adds a configuration to the concept map.</bh:Description>
        <bh:Description xml:lang="en" ls="2,3,4">Adds a configuration to the model.</bh:Description>
        <bh:Description xml:lang="en" ls="5">Adds a configuration to the expression fragment, expression
        scenario, or conditional model fragment
        </bh:Description>
        <bh:Description xml:lang="en" ls="6">Adds a configuration instance to a scenario or model fragment.
        </bh:Description>
        <skos:broaderTransitive rdf:resource="add_action"/>
        <bh:accessibleFrom rdf:resource="view_editor_ls1"/>
        <bh:locatedIn rdf:resource="configuration_property_dialog"/>
</rdf:Description>
```

Figure 7: Example of "How To?" representation

2.3.3.3. "Why?"

The "Why?" functionality communicates how the changes that are visualized in the simulation environment can be explained in terms of the simulated behavior. The "Why?" question can be asked with respect to a changing derivative value, a changing magnitude value, and a state change in the state graph.

The changes in the derivative and magnitude values are explained in terms of the values they had in the previous state and the influences and proportionalities that influenced it. See Figure 8, for instance. When asking a "Why?" question upon selecting quantity *Amount* with magnitude *Zero* and derivative *Increasing*, the communicated explanation is: "*Amount* was steady, but is positively influenced by *Flow*, which has magnitude *Positive*. Therefore, *Amount* is now increasing." An explanation can consist of an arbitrary number of influencing and propagating relationships:

- **Explanation for one incoming influence:**
  - `<bh:isInfluenced sign="SIGN" rdf:resource="QUANTITY" magnitude="VALUE"/>`
  - `<bh:previousMagnitude rdf:resource="VALUE"/>`

- **Explanation for one incoming propagation:**
  - `<bh:isPropagated sign="SIGN" rdf:resource="QUANTITY" derivative="VALUE"/>`
  - `<bh:previousDerivative rdf:resource="VALUE"/>`

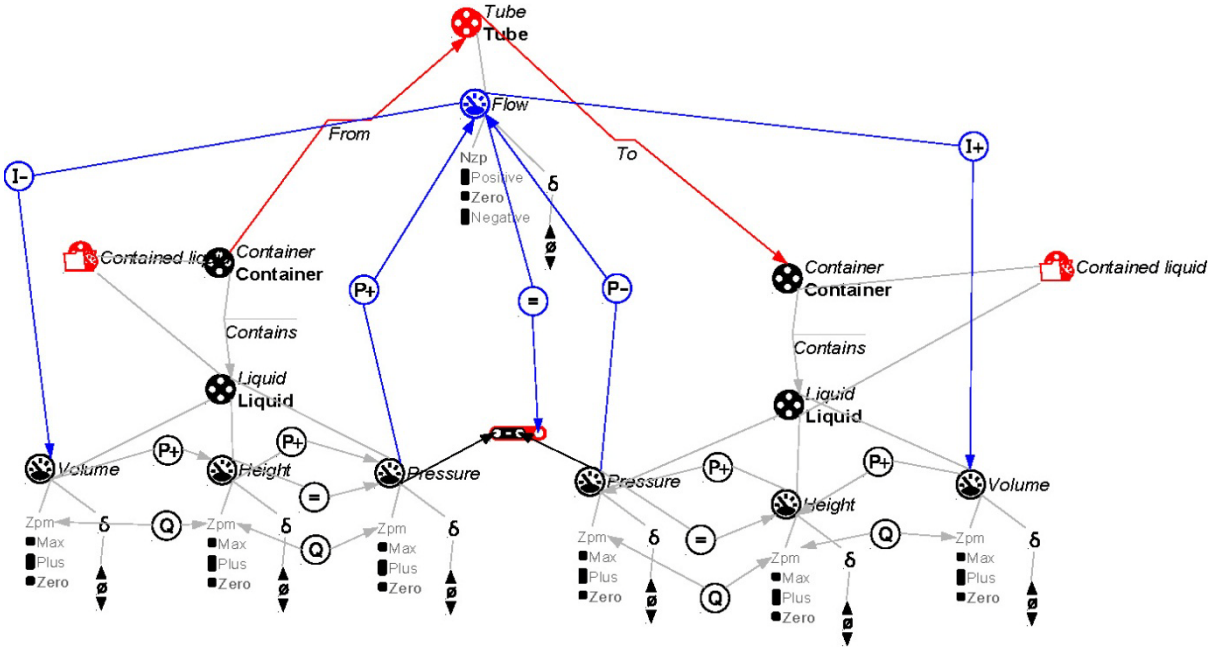

Figure 8: An example model for which "Why?" requests are formulated

The changes between consecutive states in the state graph are more complicated, because now inequalities and calculi between quantities can play a role as well. For instance, in state 1 the *Flow* between the two *Containers* in Fig. 3 is *Positive* and decreasing. In state 2 the *Flow* between the two *Containers* is *Zero* and steady. In order to explain this, the calculus between the two *Pressure* quantities needs to be communicated. The *Pressure* on the left is larger than the *Pressure* on the right. But in state 2 the two *Pressures* are equal, making the *Flow Zero*. Such an inequality change is represented as follows:

```
<bh:fromGreaterToEqual arg1="Height_Left" arg2="Height_Right"/>
```

This change combined with the proportionality information regarding the positive influence of the *Height* on the *Pressure*, gives the reason for the complex change between state 1 and 2.

2.4. Dialog Management

In this section we will explain how we used the different parts of dialog management introduced in Deliverable 5.2 "Basic tutorial tactics for virtual agents" in the Basic Help use case.

2.4.1. Sceneflow and Example Scenes

The dialogs that the Teacher character performs for the Basic Help are governed by sceneflow depicted in Figure 9.

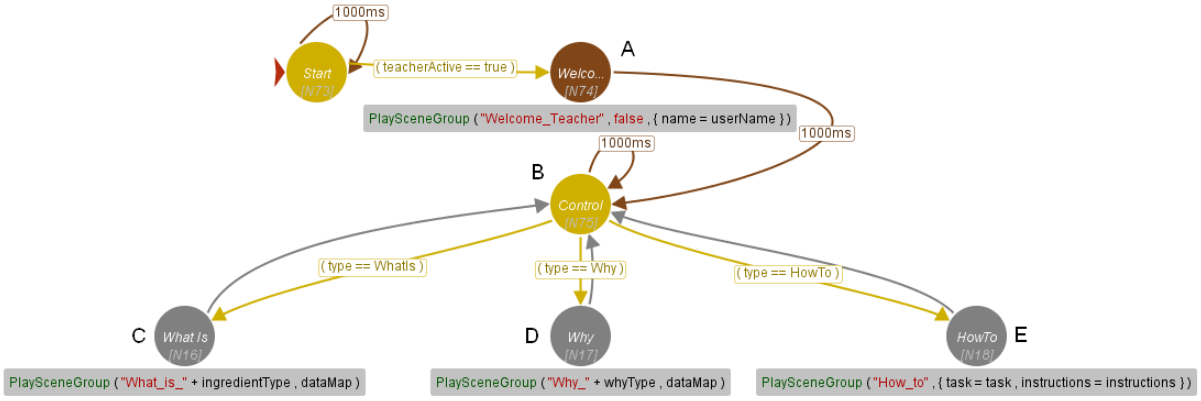

Figure 9: Sceneflow for Basic Help

After the Teacher is called by a learner, it gives a short welcome (marked "A" in Figure 9) and then waits for a help request (B). Once there is one, its type is checked and an appropriate scene is called (C, D or E). After the help is given, the Teacher waits for the next request, which could either be a follow-up question or an entirely new request.

Scenes for What Is? questions are distinguished based on the type of ingredient that learners ask about. For example, a scene used for explaining an entity might look like this:

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 quantities: $ListOfQuantities
It is part of the following configurations: $ListOfConfigurations

Note that the content of the placeholders is not specified directly when calling the scene. Instead, a map with placeholder-content pairs is supplied.

Scenes for Why? questions work in a similar way and are distinguished based on what the questions was asked about (a state, a derivative or a magnitude).

Finally, scenes for How To? questions are even easier to handle, as there is no special distinction necessary. They are simply supplied with two placeholders, one for the task that the learner asked about and one for the instructions how that task can be accomplished. Here is a very simple example of such a scene:

Scene_en: How_to
T: To $task, you have to $instructions.

2.4.2. Usage of Verbalization Module

Since all the conceptual knowledge communicated for the Basic Help is represented in a RDF-based way as described above, we first parse it using the OWL API [1]. After the necessary data is extracted, we can use it to fill the placeholders in the scenes used by SceneMaker. As an example, let us take another look at the above mentioned example "What is Water right?". If the learner would ask this question, the RDF-representation listed in Figure 10 would then be send to the Virtual Characters from the CM module.

```
<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 10: Example of an answer represented in RDF

The representation is then parsed. After that, we can fill the placeholders in the scene shown in the previous section (What_is_entity).

2.4.3. Usage of User Model

As of now, the Basic Help does not make use of the information of the user model. However, we plan to use the data provided by the user model to filter the information given in all three versions of the Basic Help in order to avoid giving experienced learners unnecessary information and do make sure inexperienced users are offered every help they need.

3. Teachable Agent

3.1. Introduction and Purpose

The design of the Teachable Agent is based on the principle of Learning by Teaching. Learning by Teaching is an effect that many students experience while learning with fellow students or when teaching younger students. They have to find different ways to explain their knowledge so that the other students understand as well. They gain different perspectives on the same subject matter and can fill their knowledge gaps. It may also be motivating to see how your teaching efforts result in the growing knowledge of your students.

A Teachable Agent (TA) transfers this principle to the field of virtual learning environments. According to the research by Blair et al. [2], a TA has a knowledge representation that can be created by the learner. From this structured knowledge the agent can extract answers to questions asked by the learner. It also must be able to explain its train of thought, so the learner can see how causal chains arise in his own model. By testing the agent's understanding of the matter through questioning, the learner can evaluate his own presentation of the knowledge and detect mistakes when the agent doesn't answer as expected.

Constant verification of the own understanding is an important part in the learning process that unfortunately often comes short due to the learners' aversion to tests. However, learners are less restrained in confronting an agent several times with the same test than in retaking this test themselves. That's why we allow the learner in our application to take part in a quiz and to send his personal teachable agent to this quiz in his place. Since the TA's knowledge mirrors an image of the learner's knowledge, he may serve as a proxy in an educational quiz.

3.2. Learning by Teaching with a Teachable Agent

The essential ideas for a teachable agent were described by Blair et al. [2] who implemented an application were learners could design a concept map to model knowledge and were supported in this task by a virtual agent. This agent used the concept map to form answers to questions asked by the learner and was able to explain his train of thought. In this way the learner could validate his model. Blair et al. also noticed that the learners tend to adopt the way the agent builds up causal chains in the knowledge model. So the agent also acts as an ideal when it comes to teaching complex reasoning. In this scenario the learner does the teaching by modeling the knowledge of the agent and asking him questions. But he learns from the mistakes the agent does as they indicate errors in his representation of the topic and also by watching the agent playing his role as a student.

Early versions of Betty's Brain, as the application later was called, only included the modeling environment for the concept maps and the teachable agent. Later versions introduced a second agent, the so called "Mr. Davis", who is best described as an assistant teacher for the learner [3]. He can quiz the teachable agent, providing a series of generated questions that help the learner to evaluate his model as a whole. But this quiz is not animated and only presented in the shell of the application. Additionally Mr. Davis provides general advice for the learner.

We detached the services of this assistant teacher character from the teachable agent and integrated them in other characters that also may provide services not related to this use case. Further in this deliverable we will focus on our implementation of a teachable agent.

3.3. Interaction Flow and Examples

The learner can choose between a female and a male character (see Figure 11). To give it more personality he can name his teachable agent by himself or just give it a randomly generated name. The name of the TA will be used in dialogs with other agents for a more believable performance.

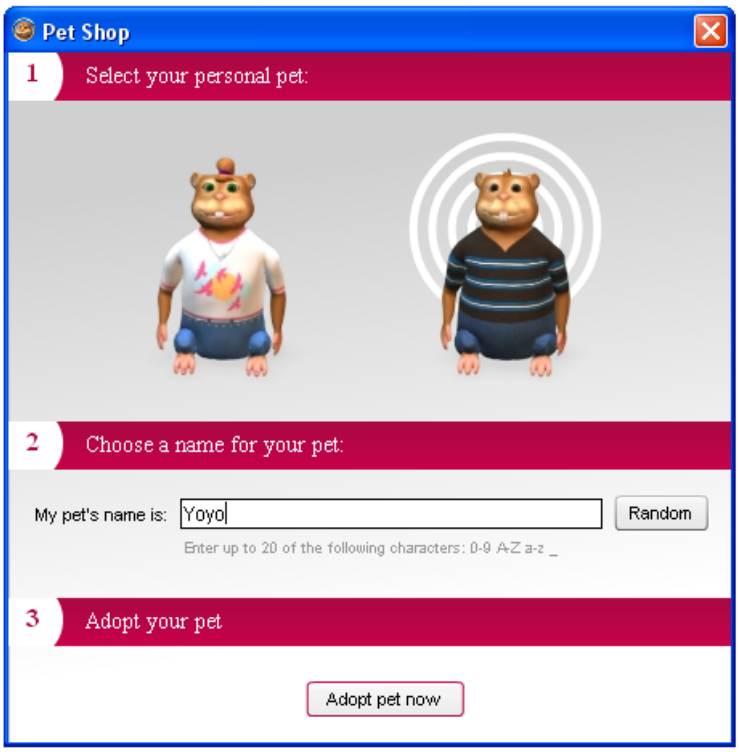

Figure 11: Selecting and naming a Teachable Agent

When the learner has decided on the appearance of their teachable agent and its name, the virtual character appears and establishes first contact with a friendly greeting (see Figure 12). Actually this step is important for the dialog management of every virtual character as this is an essential part of human everyday conversations. After the agent has introduced himself the learner has access to its various services.

Figure 12: The TA greets the learner

The most basic services for a teachable agent (according to Blair et al.) are the use cases *Ask* and *Explain*. The learner asks his virtual companion a question and may ask him to explain his answer. These questions relate to the model the learner created in the CM and are created via a menu (see Figure 13).

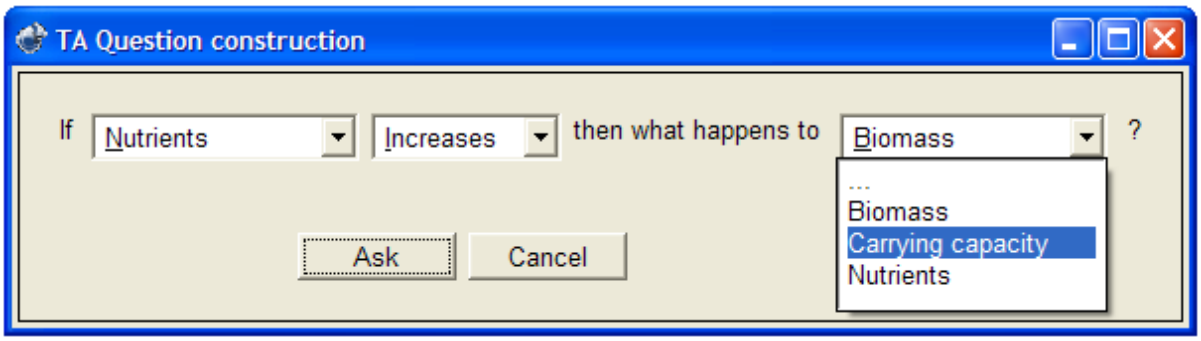

Figure 13: Dialog for asking questions

The teachable agent repeats the questions and takes some time for thinking. Of course this is only show, since the system already computed the answer. But the TA is designed to mime a student who's not smarter than his human companion. His knowledge is based on the students QR model and we want the student to think, his virtual companion has to cope with such questions the same way he does.

This behavior is motivated by studies carried out by Kim et al. [4]. They examined the effect of competency (low vs. high) and interaction type (proactive vs. passive) of virtual characters on learning success, self-efficacy and attitude towards the virtual character. Their results show that with regard to competency maximum learning success and maximum motivational effect, i.e. high self-efficacy and positive attitude, can't be achieved at the same time. High competency supports learning success, while a low-competent agent has a positive effect on self-efficacy and attitude. In our implementation of Learning by Teaching the virtual character is intended to motivate the learner to keep working on his model using the teachable agent to evaluate his work. So we designed a low-competent behavior for this agent. The learner should establish a peer-like relationship to his teachable agent in which he has the control and also the responsibility for the actions of the virtual character.

Figure 14 shows the TA answering a question.

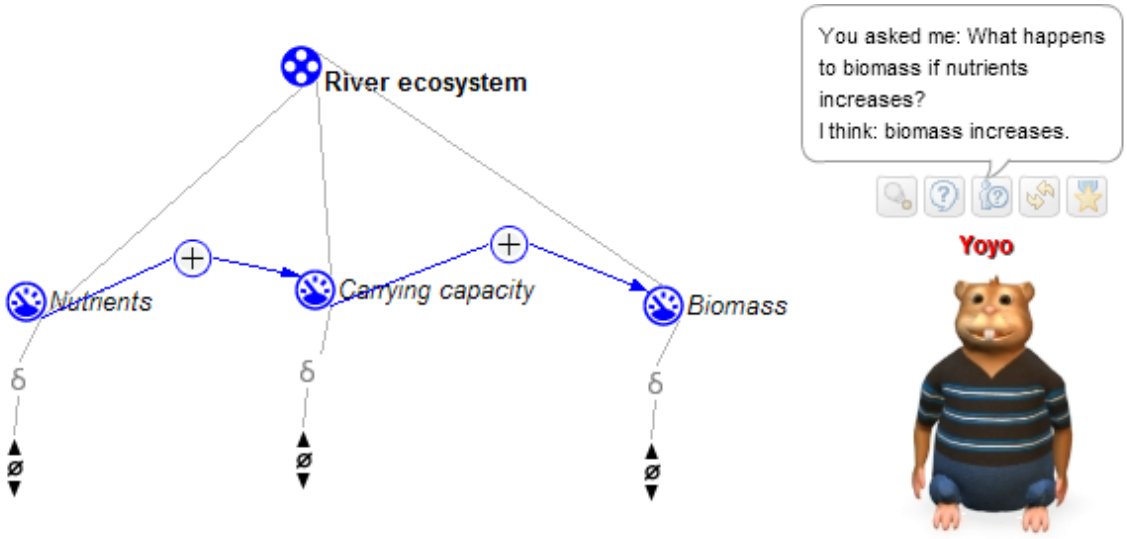

Figure 14: The TA repeats a question and answers it

If the answer of the teachable agent is right, this is a direct success for the learner, but it may be that the agent is not able to answer the question. This indicates an error in the model of the student. But also not all answers of the TA are right. It lies in the responsibility of the student to decide if the given answer is the one he expected. If this isn't the case, this might again indicate that the model is incorrect, since the answer of the TA is retrieved from it, but it may also imply that the student hasn't fully understood his own work. To clarify this case the student can ask the agent to explain its answer in detail. Of course the use case Explain only makes sense if a question was asked before. If this is not the case, the agent will tell the student that he must ask a question first. Figure 15 shows an example of the TA explaining his answer. Note the grey "history bubble" that contains its last utterance to help the learner keep track of what is being said.

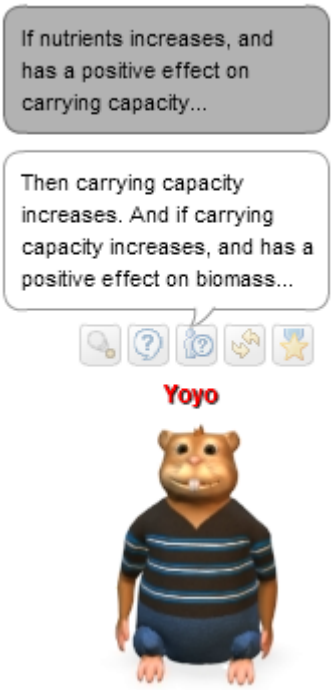

Figure 15: The TA explains its last answer

As further visual cues for the learner, the agent also makes appropriate gestures and highlight the mentioned ingredients within the model. See Figure 16 for how the TA points upwards as it is talking about an increase. Also, note the halo around the quantity it is talking about.

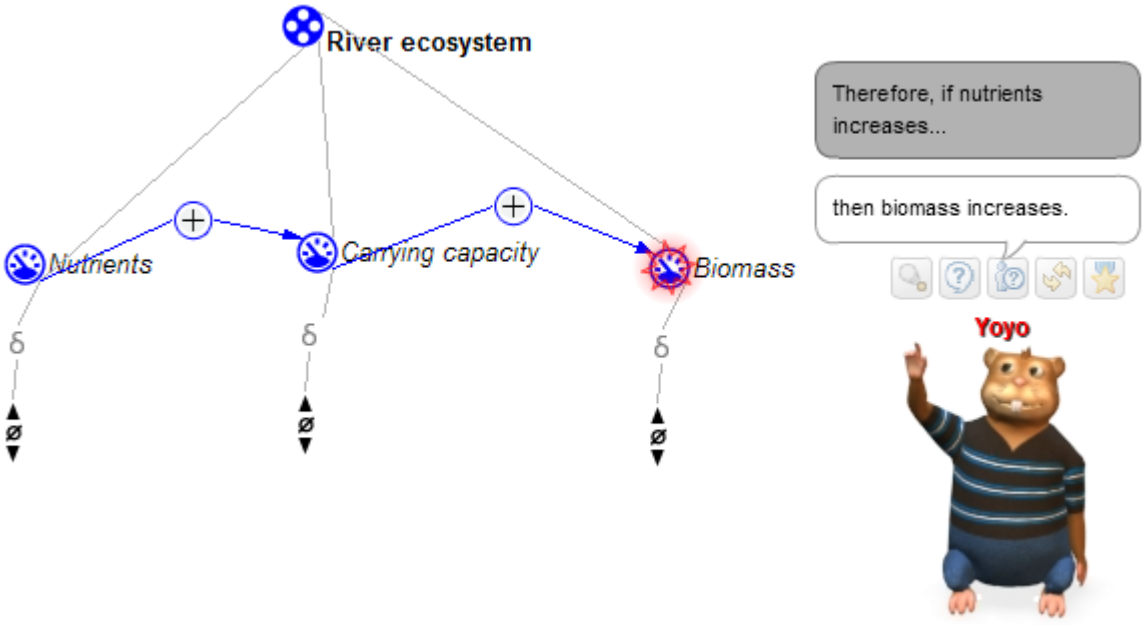

Figure 16: Additional visual cues

These two use cases cover the essential services a TA should provide to implement the principle of Learning by Teaching after the example of Blair et al. With these the student is able to receive feedback to his model by analyzing the answers his teachable agent retrieves from it.

As an additional service in our application, the learner is able to let his TA participate in a quiz. This use case is called Challenge. The TA is asked some questions by the quizmaster agent and tries to answer them correctly. Figure 17 shows a simplified version of the sceneflow that controls the dialog for the TA with which we can describe the most essential details.

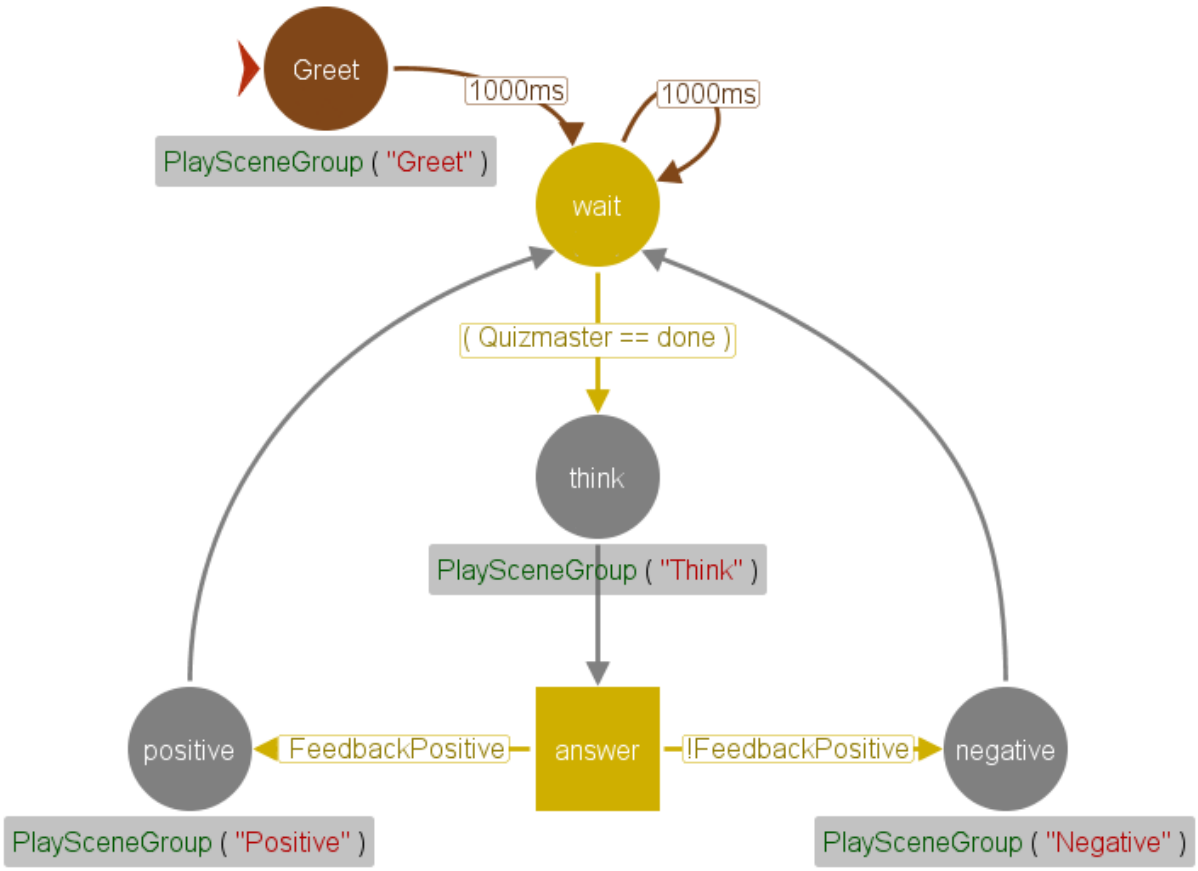

Figure 17: Schematic illustration of the Challenge use case

At the beginning of this dialog the two agents greet each other (see Figure 18). This makes the performance more entertaining and believable.

Figure 18: Quizmaster and TA greeting each other

After this greeting phase the quizmaster starts to ask his first question, as can be seen in Figure 19. During the virtual characters presentation of the quiz, the learner has the option to skip it ("Skip Dialog").

Figure 19: The quizmaster asks a question

Similar to the *Ask* use case the teachable agent will take some time to think before answering the question. The quizmaster provides positive or negative feedback depending on whether the answer was right or wrong (see Figure 20) and the teachable agent will respond accordingly with joyful or sad statements (see Figure 21).

Figure 20: The Quizmaster tells the TA its answer is wrong

Figure 21: The TA is not amused

With these responses the TA shows his interest in his results in the quiz and in this way strengthens the relation between the learner and his agent. Displaying emotions might trigger emotions like joy or compassion in the observer - the learner - that motivate him to carry on with the quiz and his work with the teachable agent. Also the dialog gets livelier in contrast to a simple sequence of questions and answers. With the response of the teachable agent to the feedback of the quizmaster the cycle ends and starts over with a new question asked by the quizmaster. After a certain number of questions, the quizmaster will present the results of the quiz, which are summarized in a table, as can be seen in Figure 22.

Quiz results

Dear Michael, in this quiz your pet Yoyo achieved a score of **40%**!

Nr.	Question	Pet's Answer	Result
1	What happens to cyanotoxins if carrying capacity decreases?	Sorry, cyanotoxins is unknown to me.	Incorrect
2	What happens to biomass if water temperature increases?	Sorry, water temperature is unknown to me.	Incorrect
3	Does biomass influence cyanotoxins directly?	Sorry, cyanotoxins is unknown to me.	Incorrect
4	What happens to biomass if nutrients increases?	biomass increases.	Correct
5	What happens to biomass if carrying capacity increases?	biomass increases.	Correct

Figure 22: Quiz results

Learners also have the option of reviewing the results for all quizzes (not just the most recent one) that their TA took so far.

The state machines controlling the behavior of the quizmaster and the teachable agent are separated and working parallel. In this use case they create a lively dialog between two virtual characters, with only a few conditions that manage the turn-taking.

With these services (*Ask, Explain, Challenge*), provided by the teachable agent, the learner is able to evaluate his QR model and his understanding of the subject matter according to the principle of *Learning by Teaching*.

3.4. Knowledge Extraction and Representation

The conceptual knowledge necessary for the Teachable Agent use case includes questions that learners can ask their TA along with the answers to them, explanations for the answers and finally questions and answers for the quiz challenge.

Details on these types of conceptual knowledge, how they are extracted by the CM and how they are represented can be found in Deliverable 3.3 "Question generation and answering".

3.5. Dialog Management

We will now explain how we used the different parts of dialog management introduced in Deliverable 5.2 "Basic tutorial tactics for virtual agents" in the Teachable Agent use case.

3.5.1. Sceneflows and Example Scenes

This section will focus on describing how agent-agent-dialogs are synchronized using SceneMaker, using the *Challenge* as an example. We will also provide examples for the other use cases *Ask* and *Explain*, but the management for these cases is much simpler. This is why we will mostly provide more details concerning the dialog content than a technical elaboration for them.

The first level of the dialog management represents the very basic structure of the teachable agent with its three use cases (see Figure 23).

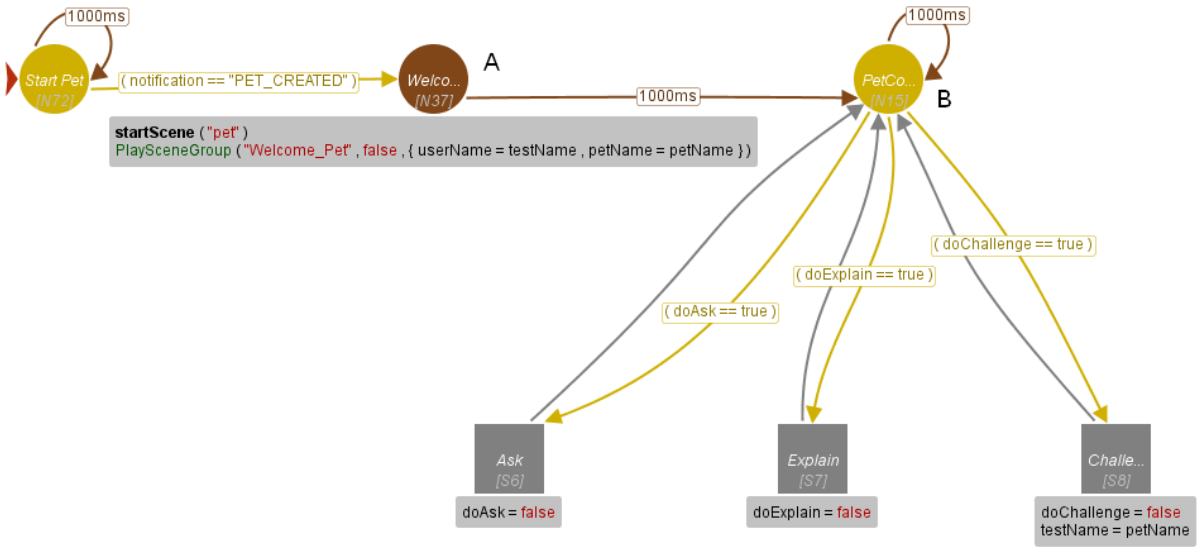

Figure 23: First level of the teachable agent's dialog flow

It also shows that the teachable agent starts with an address of welcome (marked "A" in Figure 23) after his selection via the selection screen. Here are two example scenes for this sequence:

Scene_en: Welcome_Pet
P: Hello $userName! My name is $petName. I'm your teachable agent and I'm looking forward to have a good learning session with you.

Scene_en: Welcome_Pet
P: Hello $userName! I'm $petName! I can answer specific questions and explain my answers to you. You might also challenge me and see how I do in a quiz.

The tags *$userName* and *$petName* are placeholders for the learner's name and the name of the teachable agent. They will be replaced by SceneMaker at runtime.

After this short introduction a controller node (B) handles the learner's selection of use cases. Each use case is handled in its own super node which is started according to the user's choice. *Explain* is the easiest case in regard to its state machine. Since all information the visualization needs is computed by the CM component, the dialog management has not much more to do than handle the exceptional case when no question was asked before.

Still simple but a bit more complex is the state machine for the *Ask-case*. As shown in Figure 24, the first step of the TA is to repeat the question generated by the learner via the ask-dialog (A).

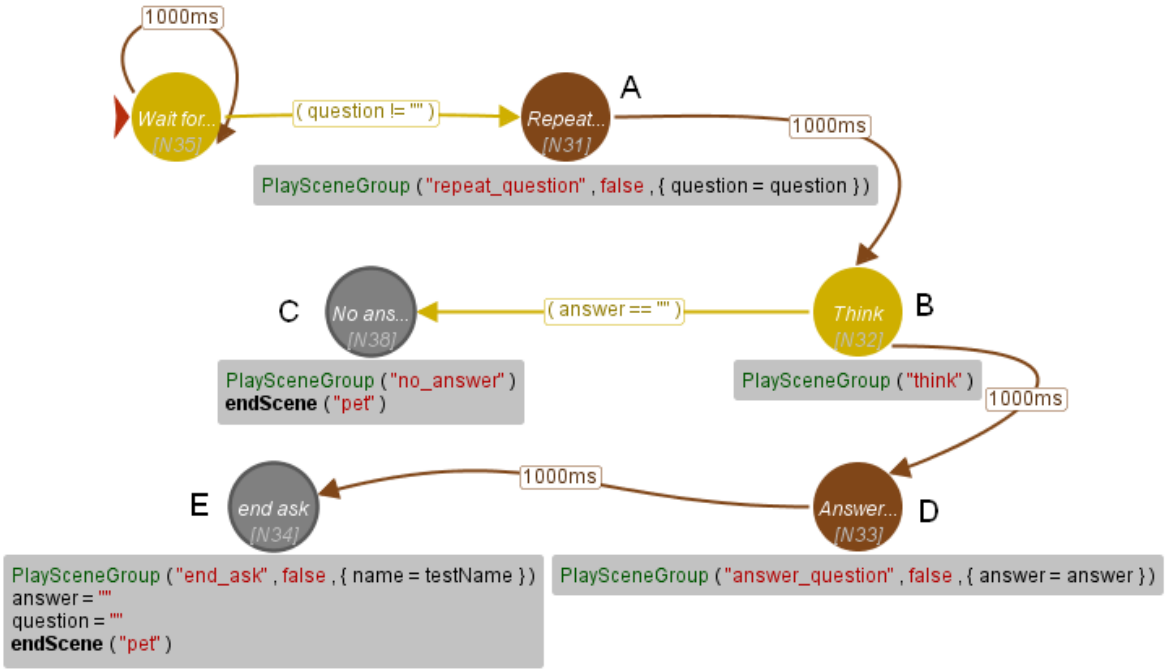

Figure 24: Sceneflow for *Ask*

The agent takes his time for thinking (B) before the state machine proceeds. If no answer has been computed, the TA will apologize for his failure (C). Otherwise the answer will be integrated in one of the following scenes and expressed by the agent (D):

Scene_en: answer_question
P: Ok, I think the answer is: $answer

Scene_en: answer_question
P: [schueler_suspense] I hope this one is right. I say the answer is: $answer

Scene_en: answer_question
P: If you ask me - and you do - the answer is: $answer

When the agent has answered the question, the dialog is ended with a short comment (E). The technical description shows that the concept of *Ask* was simple and intuitive recreated in a state machine hiding technical details in the nodes and edges.

More challenging is the dialog management for the *Challenge*-Case as the state machine for this service (see Figure 25) has to be synchronized with the quizmaster agent's part of the dialog.

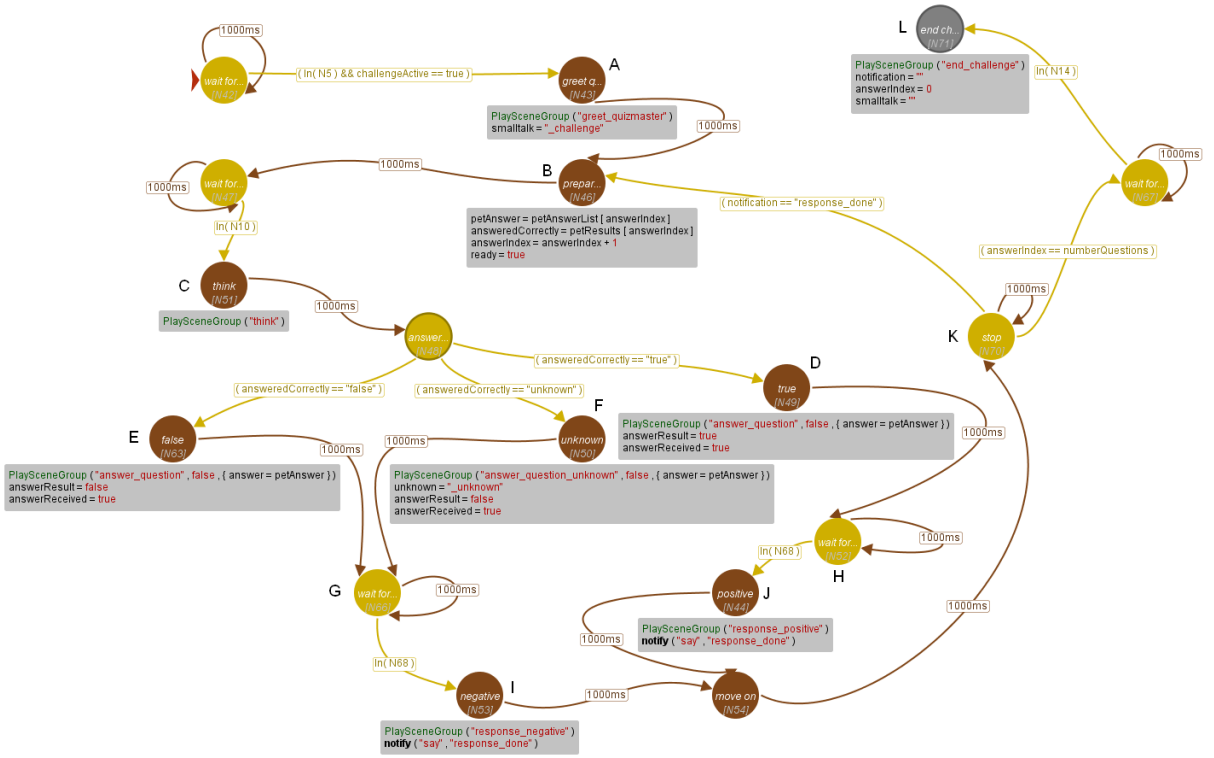

Figure 25: Full sceneflow for *Challenge*

You will notice conditions on the edges that refer to nodes that are not part of the teachable agent's state machine (N5, N10, N14, N68). These nodes are part of the quizmaster's state machine and mark the parts of the dialog where both agents start with greeting each other (A). At this point of the dialog the CM module has already computed all questions and answers, so the agents only have to integrate them into an ongoing dialog.

While the quizmaster reads his questions the TA is prepared for his answer (B). After the obligatory short thinking pause (C) the TA answers the question either correct (D), wrong (E) or tells the quizmaster he has no idea (F). As we know whether this answer is right or wrong we handle the distinction of cases for the rest of the dialog management already at this point. In each branch we wait for the feedback of the quizmaster (G, H) and let the TA then respond accordingly (I, J):

Scene_en: response_positive
P: [schueler_yes_bravo] Yes, I'm smart!

Scene_en: response_negative
P: [schueler_suspense] I'll do better next time!

This is the end of the *Challenge*-cycle that is repeated until all questions are asked and answered. Since the visualization handles all requests coming from the dialog management in a queue until this point no synchronization with its component has been done. We only took care that the state machines of the two agents send their requests in the right order. At the end of the cycle we pause it (K) so the visualization can catch up with the dialog management. For this the state machine requests a notification from the visualization component as soon as the response of the teachable agent has been uttered. Therefore we implemented the small method called `notify(type, id)`, where type defines if we want to be notified when an animation has ended or just an utterance. With the second parameter we pass an identification the visualization sends back when the action is done.

When the visualization has processed the last request, the dialog management proceeds with the *Challenge*-cycle. The quizmaster will ask his next question and the TA will try to answer it correctly. When all questions are asked and answered the agents will bid each other farewell (L) and the state machines will end the use case.

3.5.2. Usage of Verbalization Module

As of now, all the conceptual knowledge used in the Teachable Agent use case is verbalized by the CM. However, it is intended to move this to the VC, using the verbalization as described in Deliverable 5.2.

3.5.3. Usage of User Model

As this use case focuses on displaying the knowledge of the teachable agent, the user model is not used so far. Possible applications include the following situations:

- **The TA does not know the answer to a question. If checking the user model reveals that the learner also does know little about the concept at hand, the TA might make a funny comment about it.**
- **Questions in the quiz challenge for the TA might (unnoticed by learners) focus on those questions that the learners themselves have problems with, giving them additional possibilities to increase their understanding of them.**

4. Conclusion

In this deliverable we presented two use cases of the DynaLearn software, Teachable Agent and Basic Help.

The Teachable Agent offers learners a way for "Learning by Teaching", but it also represents a proxy that they can use to take tests in their place.

The Basic Help offers learners constructive advice on working with the CM in a more interesting manner than a simply online help would.

We showed what data needs to be extracted from the CM and how it is represented. But most important for the scope of this document, we showed how the dialog functionality established in Deliverable 5.2 is applied to turn this representation into engaging presentations by the virtual characters, thus enhancing the learners' experience.

5. Discussion and Future Work

While the Teachable Agent offers learners multiple kinds of interactions, it will be interesting to see how learners use them and interact with the characters in the upcoming evaluations. A part from the questions of how learners will use the existing interactions, there is also the question of whether these will be sufficient or if learners would prefer different or additional ones. For example, some learners from a preliminary evaluation stated they would like to see an explicit "Teach" interaction instead of simply adding to their model.

As a further idea for future work on the Teachable Agent the idea of extending the quiz challenge to multiple TAs comes to mind. Through means provided trough the semantic repository, learners would be able to have their agent compete with those of other learners.

Finally, as mentioned above, once the verbalization of the Teachable Agent use case is handled by the VC component, data from the User Model can be used to try to customize the interactions with the TA towards the specific preferences of a learner.

As for the Basic Help, it has also to be seen how learners react to the different kinds of help offered, how they use them and how that will influence their modeling activities.

Even more than for the Teachable Agent, the integration of data from the User Model should provide us with means to tailor the help delivered to the needs of the learners.

References

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

[2] Blair, K., Schwartz, D., Biswas, G., Leelawong, K.: Pedagogical Agents for Learning by Teaching: Teachable Agents. In: Educational Technology & Society, Special Issue on Pedagogical Agents, 2006

[3] Biswas, G., Roscoe, R., Jeong, H., Sulcer, B.: Promoting Self-Regulated Learning Skills in Agent-Based Learning Environments. In: Proc. of the 17th Int. Conf. on Computers in Education, pp 67-74, 2009

[4] Kim, Y., Baylor, A.L., PALS Group: Pedagogical Agents as Learning Companions: The Role of Agent Competency and Type of Interaction. In: Educational Technology Research and Development 54 (3), pp 223-243, 2006

e-mail: Info@DynaLearn.eu
website: www.DynaLearn.eu