Formalizing the Learner Model for CSCL Environments

Magdalena Ortiz, Gerardo Ayala & Mauricio Osorio
CENTIA, Universidad de las Américas – Puebla
Sta. Catarina Mártir, Cholula, Puebla
MÉXICO CP. 72820
{is103378,ayalasan,josorio}@mail.udlap.mx

Abstract

In this paper we present a logic formalization of a model which allows us to represent, create and maintain a Learner Model for CSCL environments in a clear and adequate manner, in order to use it as the set of beliefs an agent holds about its user, supporting the effective collaboration between the learners in the community. Our model includes a representation method for the knowledge domain and the agent’s beliefs about the learner, as well as an inference system. This system allows the agent to propose the learner adequate learning tasks, considering his own interest and possibilities and those of the other members of the community through the establishment of group-based Zones of Proximal Development. The model is formalized in Answer Sets Programming, which gives an appropriate framework for dealing with non-monotonicity in disjunctive logic programs with negation.

1. Introduction

Computer Supported Collaborative Learning (CSCL) environments consist on the establishment of virtual communities where the members learn as they collaborate in common tasks [5]. Comparing it with an ITS (Intelligent Tutoring System) a CSCL environment provides the medium for learning based on human to human interaction. In a CSCL environment, learning is a collaborative process that is made possible with the active participation and the interaction among its members, who share and construct knowledge [4]. Collaborative learning does not happen by simply using groupware tools, that’s why we need to make the most out of the collaboration and assure awareness among members of the community.

Software agents are very useful in CSCL environments, since they work in the network in order to provide their learners with personalized learning proposals and collaboration possibilities. In addition, we believe that real and effective collaboration only appears if the learners are aware of first, who is who in the network (social awareness) and, second, aware of the knowledge resources available.

In our approach, we have developed software agents that decide what to do based on their learner models. A Learner Model (LM) is a particular case of User Model consisting of a set of beliefs about the learner’s interests and capabilities [5, 25, 26]. The agent needs the learner model in order to maintain the learner’s social and concept awareness [4] and to make the best collaboration and learning opportunities available to him.

At the Universidad de la Américas, Puebla, we have previous work on learner modeling for CSCL environments based on probabilistic approaches which provided good results [22]. However, to the best of our knowledge there are no similar attempts based on logic approaches. It has been proved that many human decisions rely more on certain heuristics and inference rules than on probabilistic information, especially when it comes to having to draw conclusions with insufficient information [27]. Logic Programming allow us to complement the existing probabilistic approaches with a more human-like heuristic approach. Since declarative logic programming is widely accepted to represent and manipulate knowledge and belief systems, it seems natural to explore it in this context.

Beliefs systems for software agents deal naturally with incomplete information and non-monotonicity in the reasoning process. They can not be treated correctly in classical logic, therefore we use Answer Set Programming (ASP) as our logical framework [11].

Gelfond and Lifschitz introduced Stable Model Semantics [12] as a logic paradigm implemented by Answer Set Programming (ASP). ASP has been recognized as an important contribution in the areas of logic programming, non-monotonic reasoning and AI,
and nowadays we can find a very significant amount of research work inspired on this formalism [7, 13, 15, 24]. ASP is a well known and accepted formalism for non-monotonic reasoning. Having two types of negation, it allows us to have a direct and clear representation of the beliefs that conform our learner model. It is more expressive than normal (disjunction free) logic programming [17] and it is suitable for dealing with uncertainty.

In this paper we present the formalization in ASP of a model which allows us to represent, create and maintain the Learner Model of an agent in a clear and adequate manner, as well as to use it on behalf of the effective collaboration between the learners in a learning community. We propose the following:

a) A representation schema for the agent’s beliefs (learner model) and the domain in which the learning process takes place.

b) An inference system with the appropriate rules to derive new beliefs about the learner model and its use in order to support the effective collaboration in the community.

The organization of the paper is as follows: First we present our general approach to learner modeling in CSCL environments. In Section 3 and 4 we present in detail the different components of the formalized model. In section 5 we present some remarks about our framework in ASP and introduce DLV as a suitable implementation for some preliminary examples. Finally our conclusions and perspectives for future work are given.

2. Learner Model in CSCL Environments

In order to promote the effective collaboration in a learning community, the agent has to use its beliefs about the learner and interact with other agents in order to keep its learner aware of the interests and capabilities of the other members of the community (social awareness) [4]. We focus our attention on two main agent roles:

Proposing an adequate learning plan. Through its learner model the agent can suggest the learner the tasks in the domain which it believes to be suitable for him. For this purpose it creates zones of proximal development (ZPD) [5] according to the interests and capabilities of the learner and the other members of the community.

Helping by the establishment of workgroups. Because of the collaborative nature of the learning process, the establishment and integration of suitable workgroups according to common interests and complementary capabilities is a relevant issue. Through its LM and the interaction with the other agents, the agent can be a very useful assistant.

To be able to fulfill these tasks, we need an adequate representation of the domain (topics to be learned by the learners) and of the LM (agent’s beliefs). Then the agent will be able to apply on these two a logic derivation process through which it will infer more beliefs about the learner and, complemented with the beliefs from the other users’ agents, propose adequate learning plans and support the configuration of workgroups.

3. Formalizing the model

In this section, we present in detail the main aspects of the formalized model.

3.1 The Domain

Our domain consists of areas, tasks, situations, knowledge elements and learning resources.

a) Areas. The domain includes those areas or topics in which the community is interested.

b) Tasks. The learning process in CSCL environments implies the learners participation in solving common tasks. These tasks, as part of the domain, are goal oriented and have a collaborative nature. Tasks are goal oriented because they are planned according to preestablished learning goals. The group is intended to reach these goals through carrying out the task. Tasks must be of a collaborative nature because they are carried out by a group of learners. Each learner makes individual contributions to carry out the common task.

c) Situations. Since we consider learning as the acquirement of the capability to apply the appropriate knowledge elements in a given situation (Situated Cognition, [8]) our collaborative problem solving tasks are going to be defined in terms of situations. Each situation can occur in one or more tasks.

d) Knowledge elements. In each situation, some particular knowledge elements can be applied. According to Ira Goldstein’s approach [14] we are going to represent the relations between the knowledge elements in a Genetic Graph. This is a way of
representing knowledge which consists of individual knowledge elements connected by learner oriented links representing the evolutionary nature of knowledge. This relations (called genetic) are specialization/generalization, refinement/simplification and analogy.

e) Learning resources. We leave open the possibility to relate support material to different situations and tasks which we call learning resources. They are digital documents in the common repository.

We formalize our domain as a 2-tuple:

\[ D = < DM, GG > \]

containing a domain map DM and a genetic graph for the knowledge elements GG. The domain map is a 6-tuple:

\[ DM = < A, T, S, R, K, Rel > \]

where:

\[ A = \{ a_1, a_2, a_3, ..., a_n \} \quad n \in \mathbb{N}, \quad n \geq 1 \]

is the set of areas

\[ T = \{ t_1, t_2, t_3, ..., t_n \} \quad n \in \mathbb{N}, \quad n \geq 1 \]

is the set of tasks

\[ S = \{ s_1, s_2, s_3, ..., s_n \} \quad n \in \mathbb{N}, \quad n \geq 1 \]

is the set of situations

\[ R = \{ r_1, r_2, r_3, ..., r_n \} \quad n \in \mathbb{N}, \quad n \geq 0 \]

is the set of learning resources

\[ K = \{ k_1, k_2, k_3, ..., k_n \} \quad n \in \mathbb{N}, \quad n \geq 1 \]

is the set of knowledge elements

Rel is a set of relations. There are three types of relations in Rel:

a. Similarity relations between areas

\[ similar \subseteq A \times A \]

b. Association relations between:

- Tasks and areas \[ associated_t \subseteq T \times A \]
- Situations and tasks \[ associated_s \subseteq S \times T \]
- Resources and situations \[ associated_r \subseteq R \times S \]
- Resources and tasks \[ associated_r \subseteq R \times T \]

When we use the term associated we refer to any of these four cases.

c. Applicability relations between knowledge elements and situations.

\[ applicable \subseteq K \times S \]

The genetic graph is a 2-tuple:

\[ GG = < K, GR > \]

where K is the set of knowledge elements and GR is the set of genetic relations between them. There are three types of primitive and three types of derived relations in GR:

<table>
<thead>
<tr>
<th>Type</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive</td>
<td>analogy</td>
</tr>
<tr>
<td></td>
<td>generalization</td>
</tr>
<tr>
<td></td>
<td>refinement</td>
</tr>
<tr>
<td>Derived</td>
<td>specialization</td>
</tr>
<tr>
<td></td>
<td>simplification</td>
</tr>
<tr>
<td></td>
<td>geneticRelation</td>
</tr>
</tbody>
</table>

Within the system we represent the domain as a logic program that includes all the relations between the different domain elements, both from the GG and the DM. This relations are represented by facts, given by ground instances of the following predicates:

\[ similar/2 \]
\[ associated/2 \]
\[ applicable/2 \]
\[ analogy/2 \]
\[ generalization/2 \]
\[ refinement/2 \]

Derived relations are given by rules like the following:

\[ specialization(Ke_1, Ke_2) \leftarrow generalization(Ke_1, Ke_2) \]
\[ simplification(Ke_1, Ke_2) \leftarrow refinement(Ke_1, Ke_2) \]
\[ geneticRelation(Ke_1, Ke_2) \leftarrow generalization(Ke_1, Ke_2) \]
\[ geneticRelation(Ke_1, Ke_2) \leftarrow specialization(Ke_1, Ke_2) \]

3.2 Learner Model

The LM is the actual representation of what the agent believes about the learner’s interests and capabilities. It is the main source of information used by the agent in order to make the decisions concerning learning proposals towards more learning possibilities for the learner. For that purpose we need to have a good representation of the areas, tasks, situations and learning resources that the learner is believed to be interested in, as well as the knowledge elements which the agent believes that the learner is already capable of applying in the corresponding tasks and situations. These beliefs are expressed by instances of the predicates interested and capable as follows:

\[ interested(learner, e) \]

where learner represents a learner in the community and e represents an area, a task, a situation or a learning resource.
capable(learner, k)

where learner represents a learner in the community and k represents a knowledge element.

In ASP we have two types of negation: the classical (or true) negation and the default negation (negation as failure). Our predicates can appear as positive or classically negative atoms: ¬interested(learner, x). Under the negation as a failure schema [12] it is supposed that all the missing atoms of the predicates are false (there is no evident reason to believe they are true).

Finally, we can have disjunctions that express incomplete information. These are atoms of the form:

interested(learner, x) or interested(learner, y)

The connector or is usually called epistemic disjunction, and its interpretation is different from the traditional disjunction ∨. This type of rules is interpreted as “the agent believes that learner is interested in x or the agent believes that learner is interested in y”. Therefore it must consider both possibilities, however, the agent will not believe both at the same time unless it is forced to. The interpretation of classical disjunction, on the other side, is not related with believing, but is usually interpreted as being true at least one (often more than one) of the literals in the disjunction. These kind of atoms fall in the class of disjunctive logic programs [10], and ASP is applicable to them.

We will name the facts in the learner model basic beliefs. Using these basic beliefs, the agent can derive more beliefs about the learner, which we will call derived beliefs.

### 3.3 Deriving beliefs in the learner model

In this section, we are going to establish some logical derivation rules that allow the agent to infer more beliefs about the learner’s interests and capabilities from what it has explicitly in its LM. Obviously, it would not make any sense to start believing things with no reason. Intuitively, the ASP semantics assigns a logic program $P$ the possible sets of beliefs that a rational reasoner would hold based on $P$. [12] This rationality means that the agent will only derive beliefs consistent with its basic beliefs, and that it will not believe anything that he is not forced to believe. Here we are dealing with the concept of safe beliefs, which can characterize the semantics of ASP in terms of intuitionistic logic [21]. According to this concept, we believe those things that seems sensible to believe.

We would like to point out that the logic program $P$ that we call basic beliefs is usually known as an agent’s knowledge base. From this knowledge base, the agent can derive beliefs, that are called derived beliefs in our approach.

The derivation of new beliefs is made through inference rules, which can be defined for each domain, and according to each particular learning community. We call them belief derivation rules (BDR) Here we propose two rules that we consider general and simple enough to be valid in most cases. Despite their intuitive nature, they have shown to give interesting results. Moreover, our future work will be considering methods create better rules, like social issues of belief revision to through interaction in multiagent systems. The use of inductive logic [18] to derive rules from sample cases seems suitable and could also be considered.

This belief derivation rules set can be static or dynamically updated during the interaction among the learners in the community. In spite of believing that updating these rules could give better results, by now we simplify the model establishing only static ones.

Our belief derivation rules consider the beliefs in LM and the relations in the domain map DM. They use both types of negation, allowing us to express things like: “The agent believes that if the learner is interested in an area, he is normally interested in a similar area” or “The agent believes that if the learner is interested in a domain element, he is normally interested in the associated elements”. The rules that we propose are expressed in ASP as follows:

\[
\text{interested}(\text{Learner}, A_1) \leftarrow \\
\text{similar}(A_1, A_2), \\
\text{interested}(\text{Learner}, A_2), \\
\text{not } \neg\text{interested}(\text{Learner}, A_2)
\]

\[
\text{interested}(\text{Learner}, E_1) \leftarrow \\
\text{associated}(E_1, E_2), \\
\text{interested}(\text{Learner}, E_2), \\
\text{not } \neg\text{interested}(\text{Learner}, E_2)
\]

The basic beliefs that the agent holds, together with the beliefs derived from these rules, give each agent a learner model. Under the ASP semantics, each answer set gives a possible learner model for a learner $\text{LM}_{\text{learner}}$. Within each learner model, we will name $\text{Interests}_{\text{learner}}$ the set of all the domain elements (areas / tasks / situations / resources) that the agent believes that $\text{Learner}$ is interested in. On the other
hand, \textit{Interests}_\textit{Learner} will contain all elements that the agent believes \textit{Learner} is not interested in. Analogously, we define \textit{Capabilities}_\textit{Learner} and \textit{Capabilities}_\textit{~Learner} as the corresponding sets of knowledge elements.

Now that we have given a suitable representation of the domain and the learner model in a logic program according to the ASP semantics, let us go on to discuss how to use this learner model on behalf of the best collaboration and learning opportunities within the community.

4. Supporting collaboration through the Learner Model

Now that we have our learner model as a set of beliefs about our learner’s interests and capabilities, we are going to use them to propose the learner a suitable set of tasks. We are going to do so by supporting the creation of the learner’s zone of proximal development, ZPD. According to Vygotsky’s theory of social learning [9], the zone of proximal development (ZPD) is the distance between the actual development level and the potential development level of the learner. The actual development level is the set of knowledge elements that have been already internalized by the learner [5]. From this actual development level we can derive the knowledge frontier of each learner.

\textbf{Knowledge frontier (KF):} set of knowledge elements which the agent believes not to be in the learner’s capabilities, and which are directly related to those knowledge elements that the agent believes to be already part of the learner’s capabilities. Those knowledge elements are considered to be more easily learnt than others.

\[ KF_{\text{Learner}} = \{ k \mid k \notin \text{Capabilities}_{\text{Learner}}, \exists k' \neq k: k' \in \text{Capabilities}_{\text{Learner}}, (k, k') \in \text{geneticRelation} \} \]

In ASP, the knowledge frontier of a learner can be easily represented through the following rule:

\[ \text{kf}(\text{Learner}, K) \leftarrow \text{geneticRelation}(K, K), \text{not capable}(\text{Learner}, K), \text{Learner} \neq \text{Learner} \]

Finally, we will propose the learner a set of tasks that we consider will be appropriate for his development level, as well as interesting to her/him. This tasks will be those where the learner can apply, with the help from the other members of the group, those knowledge elements in her/his ZPD which are also related with the learner’s interests.

\textbf{Learning Proposal (LP):} The agent proposes the learner a set of tasks that include situations where the knowledge elements in the learner’s ZPD are applied and that are related to the learners interests.

\[ LP_{\text{Learner}} = \{ \text{task} \mid \text{task} \in \text{Interests}_{\text{Learner}}, \exists k : k \in \text{ZPD}_{\text{Learner}}, \exists \text{sit} : (k, \text{sit}) \in \text{applicable}, \text{sit} \in \text{associated} \} \]

The learning proposal for a learner can be inferred using the following rule:

\[ \text{lp}(\text{Learner}, \text{Task}) \leftarrow \text{zpd}(\text{Learner}, \text{KnowElem}), \text{applicable}(\text{KnowElem, Situation}), \text{associated} (\text{Situation, Task}), \text{interested}(\text{Learner, Task}) \]
5. The ASP framework and its implementation in DLV

Due to the nature of our model, ASP seems a very appropriate framework. It has several features that make it suitable:

Using incomplete information: in an environment such as this one, we can never have all the information about the learner. The success of our model depends on our ability to carry out the deductive tasks having incomplete information. Disjunctive programs are particularly suitable for such situations and they are supported by ASP. ASP is a good alternative for non-monotonic reasoning and its implementation of “safe beliefs” seems a perfectly appropriate approach. [21]

Negation as failure and classical negation: we have already mentioned how our rules are easily modeled using both types of negation. Yet another advantage of having both negations is that it allows us to deal with different levels of certainty without needing a wider truth value lattice.

Multiple intended models: The possibility of having more than one model as a result from our inference allows us to have more than one proposal for learning plans and group configuration. When we have many different proposals, we can give different priority to certain conclusions [19]. For example, when proposing a learning plan, we can prefer the tasks that appear in all answer sets (skeptical reasoning in ASP) over those that are present in only one. Another criteria for choosing models can be preferring conclusions that have a strong justification over those that are derived only non-monotonically. There are many extensions of ASP that introduce priorities and that also seem suitable in our approach [13, 24]. In our next work, we will be considering some of these extensions [20].

5.1 Implementing an example in DLV

The two best known systems that implement answer sets are DLV\(^1\) and SMODELS\(^2\). We have chosen DLV to implement the first samples of our model due to the following features:

- It includes both negation as failure and classical negation.
- It supports answer set semantics for full disjunctive logic programs.
- DLV is the only existing system supporting such an expressive language in an efficient way. It also supports integrity and weak constraints, as well as many front ends (updating, planning, preferences, etc.) that could be useful to improve the model.

Some experimental examples have been implemented using DLV with good results. They are available online\(^3\). Here we present only a small part of one of them.

5.2 Example

We have developed an example where the community learns German grammar. In this example we represent different tasks where the users have to write a conversation applying different grammar rules. Here we present a part of this domain, including only one area and two tasks. For the moment, we leave out the associated learning resources. The sets in the DM would be the following ones:

\[
A = \{ \text{area} : \{ \text{"Services"} \} \}
\]

\[
T = \{ \text{task} : \{ \text{"Eating in a restaurant"}, \text{"Shopping in the market"} \} \}
\]

\[
S = \{ s_1 : \text{"Request an article"}, s_2 : \text{"Ask the price of an article"}, s_3 : \text{"Request the bill"} \}
\]

\[
K = \{ k_1 : \text{"request_obj + bitte"}, k_2 : \text{"request_möchte/hätte (+ gern) + obj"}, k_3 : \text{"request_(bill) + bitte"}, k_4 : \text{"request_möchte/hätte (+ gern) + (bill)"}, k_5 : \text{"questionVerb + pers + obj"}, ... \}
\]

Both task are related to the only area we are dealing with. Situations \(s_1\), \(s_2\) and \(s_3\) are related to \(task_1\), and situations \(s_1\), \(s_2\) and \(s_3\) are related to \(task_2\). There are also applicability relations between our situations and the knowledge elements, for example, the knowledge elements \(k_1\) and \(k_2\) can be applied in situations \(s_1\) and \(s_3\), while \(k_1\) can only be applied in situation \(s_3\), and so on. This way the applicability relations are defined for all elements in \(K \times S\). The knowledge elements (grammar rules) are organized in a genetic graph. For example, \(k_1\) holds a refinement relation with \(k_2\), as \(k_2\) does with \(k_3\). At the same time, \(k_2\) holds an specification relation

\(^1\) http://www.dbai.tuwien.ac.at/proj/dlv/
\(^2\) http://www.tcs.hut.fi/Software/smodels/
\(^3\) http://mailweb.udlap.mx/~is103378/tesis/examples/
with $k_i$. There are also analogy relations, as between $k_i$ and $k_j$.

Given this domain, the basic beliefs in the learner models would look like the following example, corresponding to a learner named Peter:

\begin{verbatim}
interested(peter, market).
interested(peter, request_article).
capable(peter, request_bitte).
capable(peter, request_moechtingen).
\end{verbatim}

According to these basics beliefs and our belief derivation rules, the learner model of Peter would be as follows: $\text{Capabilities}_{\text{peter}} = \{ k_1, k_2 \}; \text{Interests}_{\text{peter}} = \{ a_1, t_1, t_2, s_1 \}$. His knowledge frontier would include $k_3$ and $k_4$.

Given that there is another learner in the community whose agent believes that he is capable of applying $k_3$ or $k_4$, Peter’s ZPD would also be $k_3$ or $k_4$. However, in both cases the agents give Peter as learning proposal $task_i$.

Here we present a small part of this example coded in DLV. The complete example and code are available online:

\begin{verbatim}
4 http://mailweb.udlap.mx/~is103378/tesis/examples/enc.html
\end{verbatim}
area(services).
task(restaurant).
knowledgeElement(request_bitte).
knowledgeElement(request_moechthesaette).

... refnement(request_bitte, request_moechthesaette).
specialization(request_bitte, request_bill_bitte).

... analogy(X, Y) :- analogy(Y, X).
simplification(X, Y) :- refinement(Y, X).
geneticRelation(X, Y) :- analogy(X, Y).

... applicable(request_bitte, request_article).
associated(ask_article, restaurant).

... interested(Learner, Area2) :-
  similar(Area1, Area2),
  interested(Learner, Area1),
  not ~interested(Learner, Area2).

... memberOfZpd(Learner, KnowledgeElem) :-
  memberOfKf(Learner, KnowledgeElem),
  capable(Learner1, KnowledgeElem),
  Learner <> Learner1.

... learner(peter).
interested(peter, market).

... capable(ana, request_bill_bitte) v
capable(ana, request_moechthesaette_bill).

6. Conclusions and future work

With this work we hope to contribute in the development of logic based learner modeling for CSCL environments, as well as in the practical applications of ASP. It is still under development and we hope to have more and better results in the following months. There is a strong need of logic based user models for group learning environments. We hope that this work will be an initial point in this task and bring new research efforts to the area.

Also in the area of ASP, we hope to open a new area of useful applications. The expressiveness of ASP and its many features, make it a very useful formalism in several different fields. In this case, ASP turned out to be very adequate. The modeling of learners is natural and simple under ASP and the behavior of the system is very good. For the next stage of this work, we have great expectations concerning several research projects based on the ASP formalism and how they could enrich this model. For example, for updating the LM and the belief revision process, there is a project from Alferes and Pereira [2,3] and a DLV extension for updates called LUPS, which has already used by Acosta for Belief Revision in agent systems [1]. Leite’s MINERVA project [16] can also bring interesting ideas relative to the use of ASP in agent systems.

In this paper, we have shown that ASP can be successfully applied in the area of CSCL. Our ultimate goal is to provide more evidence of the usefulness of non-monotonic reasoning and ASP as a framework for computer supported collaborative learning.

References


