GEAHS: A Generic Educational Adaptive Hypermedia System based on situation calculus

Cédric JACQUIOT
Supélec
Gif/Yvette
France
Cedric.Jacquiot@supelec.fr

Yolaine BOURDA
Supélec
Gif/Yvette
France
Yolaine.Bourda@supelec.fr

Fabrice POPINEAU
Supélec
Gif/Yvette
France
Fabrice.Popineau@supelec.fr

Abstract: GEAHS is a platform designed to ease the development of Adaptive Educational Hypermedia. In this document, we explain the underlying principles of this platform. Genericity is achieved thanks to an adaptation engine based situation calculus and RDF. This paper describes the main aspects of our system, as well as the use we make of situation calculus to create an adaptive hypermedia system.

Keywords: adaptive hypermedia, generic hypermedia, education, situation calculus, first order logic, RDF

1 Introduction

Many Adaptive Hypermedia Systems (AHS) have been developed for educational purpose in the past few years. Most of them have been created for a precise purpose, and are not reusable. In other words, these systems have to be build from scratch any time one wants to develop a new teaching platform. P. Bruzilowsky described [1] a methodology for creating an adaptive hypermedia. P. de Bra [2] studied the theoretic and generic ground that should be used for the creation of Adaptive Hypermedia Systems. An engine – AHA! [3] –, was even developed, based on AHAM specifications. Our goal has been twofold. First, we wanted to create an engine – based on principles close to those of AHAM – using standard or recommended formalisms, in order to make it as reusable as possible. Second, we wanted to provide an extensible set of reusable adaptation rules, in order to avoid as far as possible the creation from scratch of new adaptation rules. The adaptation is provided with generic built-in rules and metadata, that can be reused (and modified if necessary) by the AHS creators.

Our engine is based on situation calculus – a calculus based on first order logic (FOL) – for the adaptation and on RDF for data representation. As we shall see, situation calculus makes the adaptation problem simple to represent. RDF, as a W3C recommendation, allows us to easily import/export data to/from this language.

In order to simplify the creation of an AHS we studied in detail the situation calculus, as we think it can be used simply to create a deeply generic platform. We shall see that all processes that are not dealt with by situation calculus can easily be expressed in FOL, which is why we also worked on interfacing the different logics. Finally, we found a way to implement all the logic modules and make them work altogether.

AHS can adapt links, contents and style. The current version of our engine provides link adaptation. We intend to work on content and style adaptation in a near future.

We shall see in details the use we do of RDF and the global functioning of our system in part 2 of this document. In part 3, we will show how we use situation calculus for creating an adaptive hypermedia. Finally, we will demonstrate how the different logic elements used in our system are successfully interfaced and implemented.

2 Global architecture

This section will provide a global overview of GEAHS architecture. First we will detail the global functioning of the system, and then we will describe the choice of the formalisms and their interest.

2.1 Presentation

As suggested in [2], our system is divided into three interacting components. The first one is the learner model (LM), which provides information about the learner, for example his/her name, his/her age, his/her former passed classes… The second one is the domain model (DM), which represents information about the documents that can be recommended to the learner. The third and last one is the adaptation engine (AE), which provides – in our case – link adaptation to help the learner navigate through the documents. Let us describe the information handled by each of these modules.

2.1.1 The Learner Model

Our learner model is based on a conceptual approach of the problem. The different concepts are linked by (conceptual) relations. In order to be able to describe the semantics of the relations very precisely, our learner model is divided in two parts: the pattern and the knowledge base. The pattern lets us describe the properties of the relations, whereas the knowledge base contains the actual information about the learner.

This LM allows us to describe the properties of the relations concerning the learner, i.e. the arity of these relations, their range and domain, and possible logic rules that exists between these relations.
For example, we can provide a relation date_of_birth to allow us to describe the learner's date of birth. Our representation system allows us to define the class of this relation (learner_relation_type), its arity (2, one learner in, one date out), and the fact that it is related to the relation has_age by the rule date_of_birth(learner, date) => has_age(learner, current_date-date).

The whole purpose of this is to describe the relations' semantics as far as possible, in order to provide well-defined, reusable conceptual relations. With this model, we are able to describe what the learner knows, who he/she is, what educational methods he prefers, his examination results, global appreciations, personal information, categorization… We shall see in further detail how RDF is used for this purpose in part 2.2, and a use case in the last part of this document.

2.1.2 The Domain Model

The DM is based on the same principles as the LM. It is also split into a pattern and a database. The pattern allows us, as for the LM, to describe the semantics of the relations as far as possible, whereas the base contains the actual data about the domain. Even though we intend to provide as much freedom as possible in describing the resource's connections with one another, we also provide built-in learning objects categories (e.g.: concepts, chapters, electives…).

The resources we deal with can be documents, examples, or any kind of learning entities. The desired relations can be described in the pattern and used in the base, as long as the maximum possible semantics is given in the pattern. For example, we can describe which documents are prerequisites for others, prerequisite being a built-in relation.

2.1.3 The adaptation engine

The adaptation engine is the component of the system that adapts the path through the documents. We have decided to focus on link adaptation, content and style adaptation will be part of our future work. Our engine is based on the situation calculus, and on a set of rules to describe the required way to adapt the path through the documents.
The idea of using the situation calculus is found in [4]. As for using FOL for describing rules, the reasons are twofold. First, situation calculus is based on FOL, which makes communication between the two calculi very natural. Then, transcribing rules from natural language to FOL can be very easy. For example, if one wants to express the following rule: "A document that is read by a learner is considered to be known by this learner", one will use the following FOL rule: $has\_read(learner, document) \Rightarrow knows(learner, document)$.

In order to actually provide adaptation, we consider the learner's knowledge, his place in the learning path, and the DM, as parts of the current situation. Visiting a page, reading it or answering questions are possible primitive actions that may modify the situation. In the situation calculus, an action must be "possible" in order to be actually

executed, i.e., some conditions are required to do an action. This notion of possibility is close to its common meaning. We also use another idea from [4], which consists in including the concept of desirability in the situation calculus. The desirability is more relative to the learner -- what is good for him to get access to, whereas the possibility is relative to the physical notion of what can be done -- for example, one might access all documents if one is online, and those previously downloaded if one is offline.
As another example, visiting a document can be possible if it is available for download, but not desirable if the previous document (in a linear order) has not been scrolled yet. The rules describing the possibilities and desirabilities are FOL rules which are very similar to [5].

2.1.4 Global execution of the system

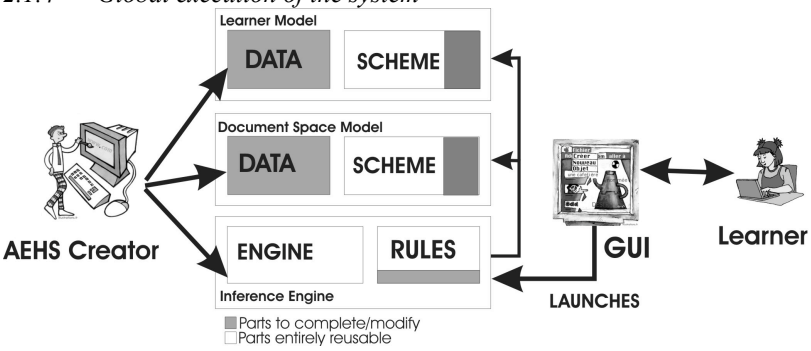

Figure 1: Global architecture schema

We will now present the way our three main components operate with one another. Fig. 1 shows that an AHS creator only has to complete the data and rules (and modify the schemes if needed) to create his brand new platform. Once the system is created, the learner can do action through a GUI. Those actions are transmitted to the adaptation engine. Using the adaptation rules, the possibility/desirability of the action is determined. In order to calculate the FOL rules, the engine interrogates the LM and DM databases.
Let us see a simple example of our system functioning (a more detailed example, showing all processes, will be given in part 4 of this document). We consider a document space made up of 4 documents (doc_i, i=1..4). doc_2 and doc_3 have doc_1 for a prerequisite, and doc_4 has doc_2 and doc_3 for prerequisites. The main rule for the user is that a document that has been read is considered to be known. The one for the adaptation is that all of its prerequisites need to be known, and that it must be a prerequisite of the objective, in order to get access to a document (i.e. for it to be possible and desirable for the learner). The initial situation for the learner is on doc_1. He/she can access doc_2 and doc_3 freely in any order from there. He/she cannot access doc_4, until he/she has read both doc_2 and doc_3. If he/she chooses to read doc_3 first, he/she can still access doc_2, but not doc_4 yet, for he/she has not read doc_2 yet -- which implies that she does not know it. Then he/she goes from doc_3 to doc_2, from where he/she can finally access doc_4. This example, which is of course very simple, shows how adaptation components provide several ways for going, as the learner could have chosen to go and read doc_2 before doc_3. Moreover, it shows how FOL rules can allow a simple description of the system.

2.2 Formalisms

2.2.1 The situation calculus

The situation calculus is based on a simple principle: applying action(s) to the current situation in order to get a new situation. The actions are possible or not in a given situation. We added the notion of desirability of an action in order to get a more refined classification of linkage possibilities, as seen in 2.1.3.

Using situation calculus makes our adaptation principle both simple and powerful. At a given time, a learner using an AHS is in a given situation: what he knows is determined; his situation in the domain is known by the system. The actions that could be done (if possible/desirable) are mainly clicking on a link that leads to another document of the domain, reading a page, or getting a test. When entering a new situation, the system calculates (using the adaptation rules in FOL) the possible and desirable link. Then, the links are provided to the learner with an adequate colouring code. According to the possibilities given to him/her, he/she will go to another document or read a page or get a test…and as soon as the new action will be done, the system will calculate the possible/desirable links again.

This shows how simple and natural this adaptation principle works. Part 3 will show in further details the situation calculus principles, as well as the way we used it for adaptive hypermedia.

2.2.2 Use of RDF

We decided to use RDF [6] to represent data and metadata for our system. We shall see the reasons of this choice before giving an example in the education domain.

The LM and DM are designed to provide easy-to-access information about the learner. This implies several constraints.

The information must be categorized. For example, one might want to know how old the learner is, and if he/she succeeded in his last computer science class. Those pieces of information are different. The following stand-alone information would not be enough to be relevant in an information research:
Learner001 age 23
Learner001 computer_scienceA01 succeeded
As a matter of fact, with this information only, the system is not able to determine if 23 is a correct age. It cannot determine if computer_scienceA01 is an element of the courses either. This is why it is compulsory for us to have a representation language that succeeds in describing meta-information about the learner. This is also right for the DM as well. Information is not relevant enough if it cannot be categorized.

The formalism of choice has to be well-known and based on simple grounds. Had we decided to use a non-standard formalism, reusability of content would not have been an easy task. Moreover, reusing information is much easier if our formalism is a standard one. This formalism must have simple grounds in order to let us build or reuse existing simple and successful tools for getting access to the information -- query languages for example.

The formalism we chose also has to be web-oriented. Because most of the information is distributed, a web-oriented formalism will be helpful. Moreover, since the content that can be provided with our system has to be reusable on line, we need our choice to allow natural distribution of content.

All these arguments convinced us to represent our data using RDF. As a matter of fact, RDF is a standard (W3C recommendation), web-oriented formalism for metadata representation. Using RDF Schema, one can easily describe the necessary metadata. RDF is already used on the web and on many web-oriented technologies, which makes it easy to import/export data from/to RDF. Finally, RDF is a simple formalism, since it is based on triples and since each element of the triple is an RDF resource.

Let us see a basic example of domain representation in RDF. We suppose that the RDF classes: ese:chapter and ese:course are defined, a course being divided into chapters. The properties has_prerequisite and is_part_of are also pre-defined. We consider one course C1 and four chapters ese:H1 to ese:H4, with prerequisites organised as in part 2.1.4. Here is the RDF representation:

```
ese:H1  rdf:type         ese:chapter     ese:H4   has_prerequisite     ese:H2
ese:H2  rdf:type         ese:chapter     ese:H4   has_prerequisite     ese:H3
ese:H3  rdf:type         ese:chapter     ese:H1   is_part_of     ese:C1
ese:H4  rdf:type         ese:chapter     ese:H2   is_part_of     ese:C1
ese:C1  rdf:type         ese:course      ese:H3   is_part_of     ese:C1
ese:H2  has_prerequisite      ese:H1     ese:H4   is_part_of     ese:C1
ese:H3  has_prerequisite      ese:H1
```

This simple example shows how easy it gets to represent a structured hierarchy in RDF, and especially a hierarchy related to education.

3 Using situation calculus [7]

3.1 Theoretical principles

Situation calculus is based exclusively on FOL, i.e. deductions can be done using nothing but FOL. As a matter of fact, GoLog implements situation calculus in ProLog, which only implements FOL deduction.

Situation calculus is a subset of FOL to which has been added the notions of situations and actions. A situation is a group of static facts that can be evaluated at a given instant. Actions are more or less complex procedures that apply

to a situation. An action can be possible or not. A set of primitive action can be given, and complex actions are built from the primitive actions (sequence, test, nondeterministic choice…).

An action `a` is made on a situation `s` leading to a situation `s'` if `a` is possible in situation `s` and is `s'` is the consequence of action `a` in situation `s`. Accomplishing a complex action results in accomplishing choices and sequences of primitive actions.

The predicates poss, do and primitive_action are predefined in situation calculus. poss(action, situation) is true iff 'action' is possible in 'situation'. Primitive_action(action) is true iff action is a primitive action. Do(action, situation1, situation2) is true iff poss(action, situation1) and action applied to situation1 leads to situation2.

In order to calculate the situation changes, the primitive action must be calculable, i.e. primitive actions must be defined. The fluents – the data describing a situation – must also be described. It is even the most important part, since calculating the next situation is achieved by calculating the next value of the fluents. The fluents' description gives us their new value from the action and previous situation. The way to calculate possibilities has to be given, since an action cannot be done if it is not possible.

When actions a1,a2…,an have been done in this order, the current situation is do(an, do(an-1,…..do(a1, s0)…)) where s0 is the initial situation. The actual content of a situation is given by the fluents' calculation. The possibilities are calculated at each step.

On top of the basic situation calculus, we have added the notion of desirability. This notion is related to what a learner really needs to know to reach his objective. For example, if several documents have the current document as a common pre-requisite, they can all be possible documents to read next. But only those who are pre-requisites of the objective are really desirable.

3.2 Adapting Situation Calculus to Adaptive Hypermedia

Situation Calculus was first introduced in order to manage robotics problems. Nevertheless, the idea has emerged in [4] and [8] that Situation Calculus could be used for adapting the web. This way, situation calculus could replace agents that are often very complex. As a robot is programmed to go from an origin to an objective using different steps, avoiding obstacles, dead-end-roads… a learner has to go from what he knows first to what his objective is, using the right path(s), and avoiding documents that are useless for his purpose.

We have worked on adapting situation calculus to our specific problem, and we found the following solution. First, we have split the possible actions into two kinds. The first kind is dedicated to the user's actions. The second kind is dedicated to the engine's actions. In a given situation, the system knows the up-to-date learner profile, the domain, and the position of the learner in the document space. This is considered to be the current situation. The system makes action. Those actions are mainly displaying the document, calculating the possible links and displaying them. Once the possible links are displayed, the user can do actions. He/she can read the document, take a test, click on a link and so on…

In order to update the learner data, we still use the situation calculus. In fact, we trigger the update rules when an action can be done and is required to be done. This shows how the inferring mechanism can be done using nothing but situation calculus. This offers advantages. First, we have a homogeneous system. Second, our system is laid on a simple principle, contrarily to many agent-based systems. Finally, this system has clearly defined semantics.

3.3 Using GoLog

Situation calculus has been implemented in ProLog. The so-made layer is called GoLog. We have modified GoLog in order to implement the notion of desirability. The program that calculates the possibilities and desirabilities is a GoLog program.

The AHS creator must define or reuse our pre-defined actions, and the way they change the situation. He/She must also define, modify or reuse the inference rules. That is all he/she has to do in order to get a functioning engine.

Then, GEAHS gets the initial situation for a given learner, and calculates the possible and desirable links for him/her. He/she makes actions that are recorded and that modify the situation, i.e. the links suggested to him/her.

This way, GEAHS's engine is laid on nothing but situation calculus, which has the many pros seen earlier. Moreover, the part in which the AHS creator is involved is quite small, which makes GEAHS a system simple to reuse for many people.

4 Logics in GEAHS

4.1 Inference rules

Rules in GEAHS are described in FOL. First, FOL has a defined semantics. Then, in some cases, translating rules from natural language to FOL is not a difficult task (as seen in part 2.1.3). Our purpose has been to allow as many people as possible to create their own rules if they are not satisfied with the set of pre-defined rules we provide.

Rules can be written in the standard form, even though it is not naturally provided in ProLog, the language we used to implement our system. In a future developpement, we intend to provide a simple interface to enter and modify the rules.

Here are the rules we decided to implement:
Possibility: It is possible to display an element if it is a link to a document and if it is not already displayed.
Desirability: It is desirable for a learner to get access to a document if the learner knows all the prerequisites of this document and if this document is a direct or non-direct prerequisite of the objective (even if the document IS the objective).

Here is their ProLog conversion:

```
%%%Possibility rules
infer_rule_poss(S,validall((t(Lien,rdf:type,element)
                & lienpage(Liens,S)
                & no (member(Lien,Liens)))
                ==>poss(affiche(Lien),S))).

%%%Desirability rules
infer_rule_good(S,validall((current_user(U,S)
                        &t(Lien,rdf:type,element)
                        & validall(t(Lien, preq, Preq)
                        =>t(U,knows,Preq))
                        & good_for_obj(Lien)
                        & no t(U, knows, Lien))
                        ==>good(affiche(Lien),S))).

infer_rule_good(S,validall((current_user(U,S)
                        & obj(A)
                        & validall(prequisite(A,B,[])=>t(U,knows,B))
                        & no t(U,knows,A))
                        ==> good(affiche(A),S))).
```

Note that `validall` replaces the universal quantifier and applies to all variables that are only inside the `validall` predicate. We have also defined a `validone` predicate that work the same way as `validall`, but for the existential quantifier.

We shall soon work on the problems of consistency and completion of the rule sets.

4.2 Logic implementation

We implemented the whole project in ProLog. First, the RDF triples are transcribed in ProLog, as triples of the `t` predicate. Then all rules are described in FOL, using our transcription from classic FOL to Horn clauses. The main mechanism is written in GoLog, which lays on ProLog.

One may wonder why we only used FOL or derivates from FOL to achieve our system. There are many reasons to this:
- FOL, as we saw, is a natural formalism for representing our problem. If many other formalisms and languages are used for solving similar problems, they all tend to reimplement some sort of FOL.
- FOL has a clearly defined semantics, which makes GEAHS easy to reuse and transform for solving other kinds of problems.
- RDF, which has been chosen as it is a web-oriented W3C recommendation, is easy to deal with in FOL. An RDF triple is very easily represented as a clause of a reserved predicate.
- Using FOL will allow us to test the consistency (are there contradictory rules?) and the completion (will the calculation end?) of a set of rules.

4.3 Global functioning of the logic modules

As we have shown before, we use several forms of FOL in our system. In this part, we want to show how we interfaced those forms.

As we implement the whole system in ProLog, we need all forms to be boiled down to Horn clauses in the end. Situation calculus was already implemented. We implemented the notion of desirability in the GoLog program. We also created a module to allow people to write logic in its standard form. This module makes standard logic understandable by ProLog. The clauses are not transcribed into Horn clauses, but dynamically analysed. The analyse is mostly instantaneous, as the main logic operators are already parts of ProLog possibilities.

The main program, written in prolog, is in charge of reading, writing and displaying data. It launches the most important part of our system, which is the GoLog mechanism. This mechanism uses rules written in standard logic, which are analysed through our logic module. All data are first transformed into ProLog clauses, which, as we already saw, is not a complex process.

In the end, all forms of logic are easily interfaced, working altogether in a simple way. We do have a homogeneous system, and this system's semantics are clearly defined.

5 Conclusions & future work

GEAHS is a powerful system which allows all kinds of people -- especially those who are not expert in computer science or software engineering -- to create an AHS for education. All formalisms and techniques used for GEAHS are either standards/recommendations, or well-defined calculus based on FOL. This allows our system to import, export and reuse data from or to other systems and formalisms. This is a fundamental aspect in today's semantic web, if we want to be able to have fully distributed documents (coming from different sources). Moreover, we provide built-in adaptation rules that are most commonly used among the various educational AHS. Those rules can be modified and completed at any time.

Moreover, GEAHS respects most recommendations of [2]. The non respected recommendations are being worked on. They mainly concern content adaptation. GEAHS is also an implementation of certain concepts for defining rules proposed in [5]. It could be used for other purposes than education.

Our future work will consist in several points. We wish to provide simple techniques for content and style adaptation. We also want to study the use of OWL, the Web Ontology Language for representing our data and metadata, since it provides pre-defined and useful relations. We also wish GEAHS to be able to determinate the consistency and completion of the set of rules on may create. Finally, after the past successful tests on small applications, we wish to finalize our GUI and test our engine on a real size problem.

6 References

[1] P. Bruzilowsky : Methods and Techniques of Adaptive Hypermedia, 1995, in Adaptive Hypertext and Hypermedia, Ed. Kluwer Academic Publishers

[2] Paul De Bra and Geert-Jan Houben and Hongjing Wu, AHAM: A Dexter-Based Reference Model for Adaptive Hypermedia, 1999, Conference on Hypertext, pp 147-156

[3] P. de Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, N. Stash, AHA! : The Adaptive Hypermedia Architecture, 2003, Conference on Hypertext, pp 81-84

[4] S. M. Ilraith, Adapting Golog for Programming the Semantic Web, 2002, Conference on Knowledge Representation and Reasonning.

[5] N. Henze and W. Nejdl, Logically Characterizing Adaptive Educational Hypermedia Systems, 2003, AH 2003 Workshop, World Wide Web Conference

[6] RDF http://www.w3.org/RDF/

[7] Raymond Reiter, Proving Properties of State in the Situation Calculus, 1993, in Artificial intelligence, vol 64, pp 337-351

[8] Richard B. Scherl and Michael Bieber and Fabio Vitali, A Situation Calculus Model of Hypertext, 1998, in HICSS, pp 205-214

[9] C. Jacquiot, Y. Bourda, F. Popineau: SEWAGO: A Generic Adaptivie Educational Hypermedia System, 2003, ACM conference on Hypertext

[10] F. Popineau, Y. Bourda, B-L. Doan: Generating Adaptive Hypermedia with Golog and Conceptual Graphs, 2003, 3rd IEEE International Conference on Advanced Learning Technologies (ICALT 2003)