GEAHS: A Generic Educational Adaptive Hypermedia System Based on Situation Calculus

Cédric Jacquiot[1], Yolaine Bourda[1], and Fabrice Popineau[2]

[1] Supélec, Plateau de Moulon, 3 rue Joliot-Curie, 91192 Gif/Yvette CEDEX, France
{cedric.jacquiot,yolaine.bourda}@supelec.fr
[2] Supélec, 2 r Edouard Belin, 57070 Metz, France
fabrice.popineau@supelec.fr
http://www.supelec.fr

Abstract. GEAHS is a platform designed to ease the development of Adaptive Educational Hypermedia, using standard formalisms. In this document, we explain the underlying principles of this platform. Genericity is achieved thanks to an adaptation engine based on 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 a simpler, more reusable adaptive hypermedia system.

1 Introduction

Many Adaptive Hypermedia Systems (AHSs) 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. Brusilowsky [1] described 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. Our goal has been threefold. First, we wanted to create an engine, based on principles close to those of AHAM [2], using standard or recommended formalisms, in order to make it as reusable as possible. Second, we wanted to provide a model both simple and powerful. We wish that as many people as possible are able to reuse our system. In order to achieve this, we also provide an extensible set of reusable adaptation rules. This way, it is possible to avoid 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. Third, we wished to prove that situation calculus, introduced in [3] for adapting the semantic web, can be applied to the problem of generic adaptive hypermedia. We also wish to show that it is a satisfactory solution for the second goal we want to achieve.

Our architecture is composed of three interacting components : the learner model, designed to represent metadata about the learner (in RDF); the domain model, designed to represent metadata about the learning resources (in RDF); and the adaptation engine, composed of a set of rules as well as a situation calculus-based infering engine, which provides the adaptation (currently, we only provide link adaptation).

In Sect. 2, we will show how we use situation calculus for creating an adaptive hypermedia. In Sect. 3, we will see how we use logic to create our architecture. In Sect. 4, we will compare our situation calculus approach to the condition/action approach developed in [4].

W. Nejdl and P. De Bra (Eds.): AH 2004, LNCS 3137, pp. 413–416, 2004.

2 Using Situation Calculus

Situation Calculus was first introduced in order to manage robotics problems. Nevertheless, the idea has emerged in [3] that Situation Calculus could be used for driving web applications. This way, situation calculus could help program agents that are often very complex.

Situation calculus [5] is based exclusively on FOL, i. e. deductions can be done using nothing but FOL. It 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 if 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.

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.

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 uses one type of action: displaying the links. This action does not trigger any rule. 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.

3 Logic in GEAHS

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. 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 with classic FOL operators, 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.

For an AHS creator, using such FOL rules can be simpler than using proprietary rules. First, if she happens to know FOL, her learning of our AH creation system will be much shorter. Then, if she already has rules written in FOL (which seems to be a natural way to express "logic" rules), she will be able to reuse them easily. Comparing these rules to other kinds of rules found in many adaptive systems, it does not seem more difficult to write FOL rules than to write rules in other formalisms. And at least these rules have a commonly-understood meaning.

We shall soon work on the problems of consistency and completion (including their tractability) of the rule sets.

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 is in charge of reading, writing and displaying data. It launches the Golog mechanism. This mechanism uses rules written in standard logic, which are analysed through our logic module.

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

4 Situation Calculus and Condition/Action

In Wu's condition/action system, the AHS creator defines a set of pairs of conditions and actions. Each time the user makes a physical action, or each time the user representation evolves, the rules' conditions are checked. If the condition is true, the corresponding action is triggered. As long as rules are triggered (possibly by other rules' consequences), their actions are applied. Rule triggering can loop. In order to be sure to achieve termination in a condition/action model, Wu and De Bra introduced restrictions to the rules that can be described using this formalism. In the end, the model they use is a transformed version of condition/action, and thus, non-standard.

In GEAHS, each time the user makes an action, the finite set of (ordered) rules is triggered. When a rule is applied, it can have consequences on other rules that come after it in the defined order, but it cannot trigger other rules. Thus, the risk for an AH creator to make up rules with no termination is reduced by the intrinsic functioning of our system. On the other hand, we did not prove yet that situation calculus is as expressive or efficient as condition/action.

5 Conclusion

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 ed-

ucation. 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).

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.

References

1. Brusilowsky, P.: Methods and Techniques of Adaptive Hypermedia. Adaptive Hypertext and Hypermedia. ED. Kluwer Publishing (1995) 1–43
2. De Bra, P., Houben, G-J, Wu, H.: AHAM: A Dexter-Based Reference Model for Adaptive Hypermedia. Conference on Hypertext (1999) 147–156
3. McIlraith, S.: Adapting Golog for Programming the Semantic Web. Conference on Knowledge Representation and Reasoning (2002)
4. Wu, H.: A Reference Architecture for Adaptive Hypermedia Applications. PhD Thesis, Eindhoven University of Technology, The Netherlands
5. Levesque, H., Reiter, R., Lesperance, Y., Lin F., Scherl, R.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming **31** (1997) 59–84