

GLAM: A Generic Layered Adaptation Model for Adaptive Hypermedia Systems

Cédric Jacquot¹, Yolaine Bourda¹, Fabrice Popineau¹,
Alexandre Delteil², and Chantal Reynaud³

¹ Department of Computer Science, Supélec, Plateau de Moulon, 3 rue Joliot-Curie
91192 Gif-sur-Yvette CEDEX, France
{Cedric.Jacquot, Yolaine.Bourda, Fabrice.Popineau}@supelec.fr
<http://www.supelec.fr>

² France Telecom R&D, 38 rue du général Leclerc,
92130 Issy-les-Moulineaux, France
Alexandre.Delteil@francetelecom.com

³ Université Paris-Sud XI, CNRS (LRI) & INRIA(Futurs),
91405 Orsay, France
Chantal.Reynaud@lri.fr

Abstract. This paper introduces GLAM, a system based on situation calculus and meta-rules, which is able to provide adaptation by means of selection of actions. It is primarily designed to provide adaptive navigation. The different levels of conception, related to different aspects of the available metadata, are split in different layers in GLAM, in order to ease the conception of the adaptation system as well as to increase the potential use of complex adaptation mechanisms. GLAM uses meta-rules to handle these layers.

1 Introduction

With the development of networking technologies, and more specifically of the internet, the number of documents available both inside and outside organizations, such as companies or universities for example, has been increasing dramatically. It is commonly agreed upon that most companies' intranets are not very efficient at providing pertinent documents to the employees, let alone to order the documents or adapt them to the user's status in the company. Many Adaptive Hypermedia Systems (AHSs) have been developed in the past few years, and can provide adaptation within an annotated corpus of documents.

The problem of adaptation in AHSs has been addressed in several ways. Generally, an adaptation engine is made of adaptation data - most often a set of adaptation rules -, which can potentially be redefined by the AHS creator in order to provide different forms of adaptation, and of an inference engine, which applies the rules in order to select documents' contents and links, and in order to update the user model, i.e. mostly his knowledge, as in [1]. Some systems like [2] provide mechanisms to check if the adaptation rules are coherent and if the use of the set of rules won't result in a bad situation - like no adaptation at all for some users, or infinite loops of rules triggering each other.

Most of these systems (cf. [2], [3]) provide adaptive navigation support and adaptive presentation, as defined in [4]. They consider fragments, atomic resources, to build up pages to provide to the user. They also modify links' visibility according to the user's model and goal.

In this paper, we focus our attention on adaptive navigation. We wish to be able to provide direct guidance, adaptive sorting and adaptive hiding. Our purpose was to create a purely declarative model, and to see how far situation calculus [5], which offers such a declarative approach, can be used as a model for adaptation in a closed-domain context [6]. Situation calculus offers a way to describe user/system interactions in an AHS very simply. Situation calculus allows us to define a model taking actions into account. An action can be "reading a document", "taking a test", "making exercises" etc. Rules classify actions according to the user's knowledge, preferences and position in the domain.

GLAM is a model for adaptation. Thus, it can be used to create different applications, in different domains. It defines a natural way to describe adaptation, thanks to the declarative aspect of situation calculus. This benefits to an AHS creator, since he has less efforts to produce in order to translate his needs into our formalism. We also provide a generic inference engine, which can take any GLAM based adaptation data into account to provide adaptation.

We also introduce a layered rule model. Usually, rules for adaptation can take into account any data relative to the user's preferences, on the one hand, and to his knowledge of the domain, on the other hand. Writing such rules can be quite complex: their premises can be numerous and of different natures. Such complex rules are hard to debug as well as to maintain, and simpler rules only offer limited adaptation. Moreover, user's preferences and knowledge are very different kinds of data about the user, thus they influence adaptation in different ways. Preferences are domain-independent. They help to choose between different kinds of documents to offer: examples, illustrations, summaries, or how fast the user can learn. On the other hand knowledge, which is domain-dependant, tells what document can or cannot be read by the user, what concept can or cannot be treated. Thus, GLAM includes a rule system layered according to the nature of the user metadata being used. The layering is achieved using meta-rules.

In section 2, we present the global architecture of GLAM. In section 3, we present situation calculus and its role in our context. In section 4 we describe our adaptation model, and we detail its layered rule system. Finally we compare our approach to other well-known approaches and conclude with some perspectives about making a full generic AHS using GLAM for adaptation.

2 Global Architecture

As in most AHSs, we consider a user model, a domain model and an adaptation model. In this paper, we won't discuss the details of the user or domain models, but we focus on the adaptation model. However, we have designed GLAM in such a way that it can handle various kinds of metadata about documents and users - including relations.

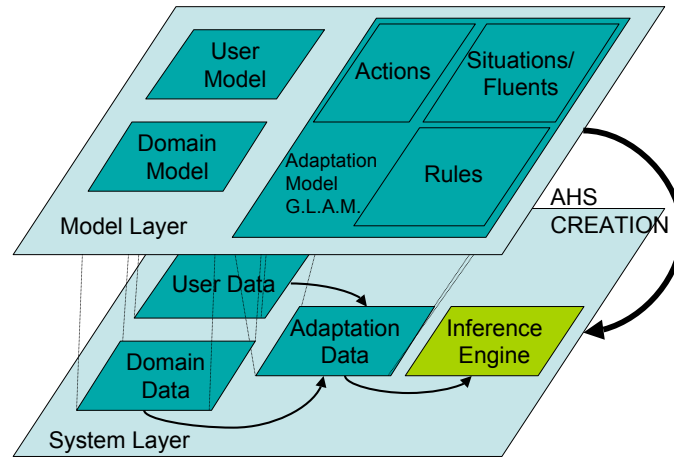


Fig. 1. GLAM's architecture

In this section, we introduce the global architecture of GLAM. The way our system works is presented in figure 1. We present our architecture according to two abstraction layers. The model layer, which provides models for implementing the data into the AHS, and the system layer, which provides the data itself and the inference engine. The data is passed to the inference engine, which is able to compute the next possible step(s). As in many systems, the inference engine is reusable, since it can work with different adaptation data.

Our adaptation model is based on Sheila McIlraith's [7] adaptation of situation calculus presented in [7], and on results specific to planning techniques [8, 9]. Situation calculus provides a simple-to-understand, well-founded and expressive-enough basis for an adaptation model. Situation calculus is made of actions, situations and precondition rules (cf. section 3).

As detailed in section 4, we have enriched situation calculus in two ways. First, we have added notions related to preconditions in situation calculus, and redefined notions found in [7]. Secondly, and most importantly, we have defined a layered rule model, which replaces and enriches situation calculus's simple rule system. Criteria relative to the user are taken into account at different levels of the rule model, in order to ease the development of a system implementing potentially complex adaptation strategies.

Our inference engine uses data, described using our own version of situation calculus, to provide adaptation : it is able to tell what actions are best suited for the user's next step, according to his profile and position in the domain.

3 Situation Calculus

3.1 Why Using Situation Calculus?

Situation calculus [5] is a logic model, based on first order logic, that allows to describe observable situations modified by actions and to reason about these

different items. Thus, our reasons for using situation calculus to provide adaptation are twofold. First, situation calculus allows to closely model reality : at any given time, situation calculus represents a situation as the result of a sequence of actions applied to an initial situation. Then, in any given situation, it is possible to do some actions among all existing actions. The fact that an action can or cannot be accomplished in a given situation is provided by rules, taking into account the current situation and the potential actions. Once an action is accomplished, the situation is modified according to this action.

In an AHS, a situation is given by the position of the user in the domain - which is the result of the various actions the user has achieved and of his initial knowledge and position - and by his preferences. An action in a given situation is to access a document that the user's current knowledge allows him to read. Once the user accomplishes an action, the situation is modified, i.e. his profile is updated. Thus, situation calculus is natively close from the reality of a user using an AHS and can easily be used to provide an adaptation model and the corresponding inference engine in an AHS.

3.2 Notions of Situation Calculus

Situation calculus is a formalism made of three distincts elements :

- Actions: they are the basis of situation calculus. They modify the situation when they are achieved. There are two kinds of actions : primitive actions and composite actions a.k.a procedures. In GLAM, we only focused on primitive actions, since the user takes them one at a time. One needs to declare every primitive action he intends to use using the *primitive_action* predicate. For example:

primitive_action(read(Document)).

primitive_action(read_about_concept(Concept, Select_Example)).

This last example shows that, if we add a document composition engine to our system layer, it is possible to achieve content adaptation on top of adaptive navigation. However, we did not address this possibility (yet).

- Fluents: they allow to observe the current situation. They are predicates defined for the initial situation, as well as for the situation resulting of an action "done" in the previous situation. The predicate *holds(Observation, Situation)* allows to describe if an *Observation* is true in a given *Situation*. The operator *do* transforms .For example:

holds(read(Document), initial_situation) ← false

holds(read(Document), do(read(Document), Previous_Situation)).

holds(read(Document), do(read(Other_Doc), Previous_Situation)) ←

holds(read(Document), Previous_Situation).

means that no document has been read in the initial situation. A document is read if the last action that was "done" consisted in reading it or if it was already read in the previous situation.

- Preconditions: the *poss* operator is used to define rules that state if an action is possible or not in a given situation.

More details about situation calculus can be found in [5] and [10]. Sheila McIlraith [7] added a finer level of precondition by defining the notion of desirability. It is then possible to accomplish an action if the resources are available, and desirable to do it if the user's knowledge is sufficient. Josefina Sierra-Santibañez [9], who studied situation calculus as a way to achieve heuristic planning, refines this model by introducing a notion of order among the actions.

4 Modifying the Situation Calculus Preconditions to Match AHSs Needs

In GLAM, we decided to reuse the notions of primitive actions and fluents without redefining them. As one can define the actions and fluents he wants in a declarative manner, there appeared to be no need to redefine those notions in our context. However, preconditions are treated in a very simple manner: if a precondition rule is true, the corresponding action is possible, otherwise it is impossible. Moreover, they are described by a rule system using horn clauses, which is not especially made for AHSs (cf. 4.2). Thus we decided to redefine the notion of precondition to make it fit better in AHSs.

4.1 Redefining the Notion of Precondition

In order to redefine preconditions in situation calculus, we took ideas from [7] and [9]. We decided to use the notions of desirability and preference. These notions allow to create a double classification among actions. First, some actions are desirable, some are possible and some are bad. Secondly, within any of the previous categories, we can establish an order among actions, potentially allowing us to guide the user in a step-by-step manner. However, since we work in a closed-domain context, we did not use the same definitions as those proposed in [7]: the notion of document availability is not very useful in this context. Thus, we decided to redefine these notions so that they provide indications about the action's level of interest within the adaptation model. The notions of possibility, desirability and order are redefined as follows:

- It is possible (predicate **poss**) to accomplish an action if the user is able to understand the document that will be presented to him as a result of this action.
- It is desirable (predicate **good**) to accomplish an action if the user is able to understand the document that will be presented to him as a result of this action, and if this document leads to his goal(s).
- It is better (predicate **better**) to accomplish an action than another if it is more adapted to the learner's preferences than the other.
- It is bad to accomplish an action if it is neither desirable, nor possible. Thus, there is no predicate associated to bad actions, avoiding potential conflicts among rules.

4.2 The Layered Rule System

Layering the Adaptation Rules. AHSs can usually adapt their behavior to different aspects of the user : preferences, knowledge, goal [3, 11]. Rules in AHSs are often a mix of several different aspects [3]. In our version of situation calculus, one can describe rules for selecting good actions, possible actions, and actions better than other actions. These rules are triggered after each action is achieved, in order to compute the next possible one(s). They are independent. They cannot trigger each other. Even though they are triggered in a given order, it has no influence on the result, which is a partially ordered set of actions. Preconditions only help computing the category - bad, possible, desirable, better than - to which an action belongs. However, as in many systems, if one wishes to take several aspects of the user into account in a given rule, he has to add premises to the rule. This can lead to quite complex rules, where it becomes difficult to distinguish the different aspects of the user taken into account for adaptation. Maintaining such a rule system can be tedious.

In order to address this problem, our idea was to "adapt the adaptation" according to the user's characteristics. To do so, we define two levels for rules. First, our base rules only use data about the domain and the position of the user in this domain, i.e. domain-related knowledge. They describe different adaptation strategies, potentially incompatibles. Then, the inner characteristics of the user, like learning preferences, learning speed for example, are taken into account at the meta-level. These characteristics are domain independant, i.e. they can be reused in several domains. A set of meta-rules uses the user's characteristics to select the adaptation base rules which will be used to provide adaptation for this user. For example, some base rules can describe a linear strategy, some can add the necessity for the user to practice exercises, some rules can also indicate that the fastest way to the goal is the best way. Rules recommending exercises and rules that help to achieve the goal as fast as possible will be mutually exclusive. Thus, the meta-rules will help select the correct rules for the current user according to his profile.

This approach offers several advantages. First, the relations between the user and the domain on one side, and the inner characteristics of the user on the other side, which are often separated in the user models, are no longer mixed in the rules. They are separated at different levels of the adaptation model. The base rules provide a mean for action selection in a given position in the domain, whereas the meta-rules select the best set of adaptation rules for the current user. Maintaining the system becomes simpler since base rules and meta-rules are smaller, and thus easier to understand for the AHS creator, than classic base rules. Finally, this way of selecting rules is much finer than a classic stereotyped approach [3], since it is possible to mix up numerous characteristics about the user, and every different set of characteristics can provide a potentially different set of adaptation rules to use in the inference engine.

A Meta-rule Model. Our meta-rule model is an adaptation of a model created by Jagadish [12]. This model was originally designed for automatic database

updates. It offers possibilities to help a system creator: it uses efficient algorithms to control the properties of the meta-rules.

It offers a formal approach for selecting rules according to their premises. In [12], a rule is said to be fireable if, and only if, all its premises are true. Meta-rules allow to select a subset of the set of all fireable rules, called the execution set. This set is generated using four kinds of meta-rules. Each kind of meta-rule is a binary relation between rules. The first kind of meta-rule is called requirement meta-rule, and allows to express that a rule requires another rule to be selected for execution, in order to be in the execution set. The second kind is called exclusion, and allows to describe what pairs of rules cannot be in the execution set together. The third one is called preference, and allows to describe which rule to prefer over which other rule when the two rules are exclusive from one another. Finally, the order meta-rules provide a total or partial order among the rules in the execution set.

The system also provides axioms that explain how to deduce new meta-rules from a set of given meta-rules, to detect determinism and order issues, and to select the execution rules. Thus, they can help the system creator by detecting flaws in his set of rules. For example, if a rule is exclusive from itself, the system can inform the creator that it is useless. The system can also detect cases where two rules are exclusive from one another, and yet there is no way to prefer one rule. This is called the determinism problem, and can be checked in polynomial time. If a total order among selected rules is required, the fact that the order meta-rules provide a total order for all possible fireable set can also be computed in polynomial time. All those verifications only depend of the meta-rules. They do not depend on the base rules.

Our AHS-oriented meta-rule System. We modified this system by changing the notion of fireability, which badly fit with situation calculus, and by incorporating account AHS specific notions, related to the user's capacity.

For AHSs, we define our rule system as a tuple $\langle V, F, M, \delta \rangle$ where :

- V is a set of base adaptation rules, which take the form of horn clauses whose results are degrees of desirability for actions. These rules take into account the current situation and a potential action, and determine if the action is desirable (or possible, or better than another one). For example:

$$\text{good}(\text{read}(\text{Document}), \text{Situation}, \text{User}) \leftarrow$$

$$\text{not}(\text{read}(\text{User}, \text{Document})), \text{good_for_partial_goal}(\text{Document}),$$

$$\text{current_document}(\text{CDocument}),$$

$$\text{prerequisite}(\text{CDocument}, \text{Document}).$$
means that it is desirable to read a document if it has not already been read, if it leads to the goal and is one of the documents directly linked to the current document by a prerequisite relation. The only premises allowed for base rules are domain-related relations.
- F is a set of firing criterions. These criterions are related to the user's inner characteristics. For example : $\text{fast_learner} = \text{true}$.
- M is a set of meta-rules. We reused the four kinds of meta-rules introduced in [12], but our meta-rules are between sets of rules instead of single rules,

allowing us to give information about all rules related to a specific user's characteristic at the same time. For example:

$$\{rule1, rule2\} \supset \{rule4, rule5\}$$

means that rules 1 and 2 require rules 4 and 5 to be selected for execution in order to be in this execution set.

- δ is a function that associates base rules with firing criterions. Unlike in [12], we cannot use the premises of a rule in a given situation as a firing criterion, since a rule is used several times in a give situation, according to the action that is being tested. δ allows to compute the γ function, which associates every criterion with the set of rules fireable according to this criterion. Thus, it is possible to write meta-rules about sets of rules related to a specific criterion, i.e. to write meta-rules about requirement, exclusion, preference and order between user's criterions. For example:

$$\gamma(\text{fast_Learner} = \text{false}) \supset \gamma(\text{need_exercise} = \text{true})$$

means that rules for slow users will include rules providing exercises.

Once those four elements are described, the part of the inference engine dedicated to the rule processing uses the axioms in [12] and formulas derived from these axioms, which make deductions about our meta-rules, to generate all deductible meta-rules. It checks if some rules will never be selected, if the system will always select the same set of base rules for a given set of criterions, if all rules that can be selected simultaneously are totally ordered, and if some set of criterions can lead to an empty set of adaptation rules. These verifications are intended to help the AHS creator who creates a GLAM-based AHS. They all are in polynomial time depending on the number of meta-rules. Providing meta-rules for sets of rules instead of single rules eases the creation of the system as well as it helps debugging it, since the meta-rules for set of rules are less numerous than meta-rules for single rules.

After checking all those properties, the system is ready to use. For each user, it evaluates the different firing criterions. It pre-selects the "fireable" rules related to this criterions. Then, it looks at the meta-rules to see if all rules can be put in the execution set or if some must be withdrawn, e.g. if all their prerequisites are not in the fireable set, or if they are exclusive of other preferred rules. Once the execution set is computed, it is ordered.

The decomposition of our rule model in four different parts makes it clearer to distinguish the different categories of elements to take into account. It limits the potential mix up of premises in the base rules as well as their number. Finally, it allows to add new base rules without modifying the meta-rules, thanks to the associations between rules and criterions.

5 Related Work

In this section, we discuss different approaches and how our system differs.

In AHAM [11], like in many other systems, the adaptation model relies on condition-action rules. Theses rules provide adaptation as well as user model update. Condition-action rules can trigger each other, and thus often need to

be restricted to ensure termination and confluence. On the contrary, in our rule system, the nature of rules prevents us from termination problems, and we implemented a verification algorithm which can check if the system is deterministic in all cases, i.e. confluent.

The adaptation system in [13] uses TRIPLE rules for directly interrogating the user and domain data. These rules can recommend documents, which is close from action selection. Moreover, the recommendations can be more or less strong, which is close to our notion of possibility/desirability/preference. However, the layering in our system allows to separate different semantic levels in our data: preferences and knowledge are dealt with separately in GLAM, in order to simplify the conception of the system. By using distinct rule levels, we prevent an AHS creator from writing very long rules, with many premises. This layering is not arbitrary: it lies upon an intrinsic semantic separation of the nature of data manipulated to provide adaptation. Moreover, TRIPLE rules can only manipulate "subject+predicate+object" statements, whereas G.L.A.M. rules can manipulate other data representation formalisms.

SCARCE [3] uses stereotypes to select the kind of adaptation to provide. In LAG and LAG-XLS [14], user-related preferences allow to select an adaptation strategy, for example, to adjust presentation of selected concept to present to the user. Our rule system goes beyond stereotypes or selection of strategy. In GLAM, one describes relations between groups of rules associated to one or more user criterion(s). These relations allow to describe the necessary rules for a strategy without having to select them manually. Moreover, a strategy can be based on several user criterions. If one takes 5 binary criterions into account, the number of potential strategies is $2^5 = 32$, which means that without meta-rules, one would have to describe 32 coherent groups of rules manually!

6 Conclusions and Future Work

In this paper, we introduced GLAM, an adaptation model for closed-content domain AHSs. We have already implemented a prototype system using this model, which is able to provide adapted navigation. We implemented the verifications for the rule system. They allowed us to check if our examples were correct and to ease the debugging process.

Using situation calculus as the core for adaptation allowed us to have a declarative representation of the system as close as possible to reality, hopefully easing the reusability of GLAM in many potential contexts. Our layered rule system introduces a way to describe adaptation by selecting adaptation strategies.

Thanks to the form of its rules, GLAM can take into account any kind of relations in the user and domain models. These models can be very different, since the different properties and relations can be different according to the domain. We now intend to provide meta-models for the user model as well as for the domain model. These meta-models are meant to help generating models fully compliant with GLAM. They will also offer reusable relations and metadata, in order to ease the creation of AHSs.

References

1. DeBra, P., Houben, G.J., Wu, H.: AHAM: A dexter-based reference model for adaptive hypermedia. In: UK Conference on Hypertext. (1999) 147–156
2. DeBra, P., Aerts, A., Berden, B., de Lange, B., Rousseau, B., Santic, T., Smits, D., Stash, N.: Aha! the adaptive hypermedia architecture. In: UK Conference on Hypertext. (2003) 81–84
3. Garlatti, S., Iksal, S., Tanguy, P.: Scarce: An adaptive hypermedia environment based on virtual documents and semantic web. In: Adaptable and Adaptive Hypermedia System. (2004) 206–224
4. Brusilovsky, P.: Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction* **6**(2-3) (1996) 87–129
5. Reiter, R.: Proving properties of state in the situation calculus. *Artificial Intelligence*, 64(2):337-351 (1993)
6. Jacquot, C., Bourda, Y., Popineau, F.: Reusability in geahs. In: Proceedings of Workshops in Connection with the 4th International Conference on Web Engineering, Rinton Press (2004) 199–209
7. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: roceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02). (2002)
8. Fikes, R., Nilsson, N.: Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **1** (1971) 27–120
9. Sierra-Santibañez, J.: Heuristic planning: A declarative approach based on strategies for action selection. *Artificial Intelligence* **153** (2004) 307–337
10. Levesque, H.J., et al.: Golog: A logic programming language for dynamic domains. *Logic-based artificial intelligence* pp 257 - 279 (2000)
11. Wu, H., de Kort, E., DeBra, P.: Design issues for general-purpose adaptive hypermedia systems. In: HYPERTEXT '01: Proceedings of the twelfth ACM conference on Hypertext and Hypermedia, New York, NY, USA, ACM Press (2001) 141–150
12. Jagadish, H.V., Mendelzon, A.O., Mumick, I.S.: Managing conflicts between rules. *J. Comput. Syst. Sci.* **58**(1) (1999) 13–28
13. Dolog, P., Henze, N., Nejdl, W., Sintek, M.: The personal reader: Personalizing and enriching learning resources using semantic web technologies. In: Proceedings of the third International Conference on Adaptive Hypermedia and Adaptive Web-Based Sytems, Springer (2004) 85–94
14. Stash, N., Cristea, A., DeBra, P.: Explicit intelligence in adaptive hypermedia: Generic adaptation languages for learning preferences and styles. In: Proceedings of the HT 2005 CIAH Workshop. (2005)