Situation calculus and personalized web systems

Georges Dubus, Fabrice Popineau, Yolaine Bourda
SUPELEC Systems Sciences (E3S) - Computer Science Department
Gif sur Yvette, France
Email: georges.dubus@supelec.fr, fabrice.popineau@supelec.fr, yolaine.bourda@supelec.fr

***Abstract*—Personalized systems are a response to the increasing number of resources on the Internet, but can be difficult to create. In order to facilitate the design and creation of such personalized systems, we aim at formalizing them. The situation calculus is a logical framework that has often been proposed to model web applications and even personalized ones. However, the details of its use are much more rarely explained. In this paper we will show that it is needed to carefully consider which variant of the situation calculus to choose. We will precisely show why we want to use the so-called guarded action theories. We explain why and how it fits into an architecture. We introduce two scenarios of personalized applications to illustrate this choice.**

***Keywords*-situation calculus; personalization; web application; adaptive systems;**

I. Introduction

The Internet is quickly becoming the main way of communicating and sharing information. The amount of content available online has increased exponentially. A single website often contains so much information that a user can't grasp it all, and may even be unable to navigate to find the information he wants. Personalized systems – systems that behave differently depending on the user – emerged to provide guidance to users in such cases.

Different types of personalized systems exist to answer different problems. Adaptive hypermedia guide the navigation in a set of resources, recommender systems offer items to the user that are likely to interest him, and personalized information retrieval systems provide the user with answers to his queries that are tailored to his needs. These different systems have distinct inner workings and distinct logical models, if any. Some systems cross the borders between those types. For example, an e-learning web application may provide an adaptive hypermedia with learning resources, and also recommendations of external resources, such as a book to read to go further in a subject.

Personalized systems are not as widely used as they should be because their creation is a difficult process [1]. There are tools to help with the creation of some of the systems (adaptive hypermedia, recommender systems), but they are dedicated to one single system, and there are no general tools covering all the personalized systems. We aim at defining a general formalization of personalized systems. This will enable us to create tools to help with the design of personalized systems and to prove that such systems behave correctly, as we will see in section IV-B.

Our ultimate goal is to model personalization from a logical point of view. To deliver personalized information to a user may require a good amount of intelligence. We build our view of the problem in the context of artificial intelligence rational agents as presented by [2]. In this context, the web application is the agent, and it interacts with its environment. Users are part of the environment. The goal of the agent is to achieve some task related to the users, like browsing a complete course in e-Learning or presenting relevant news articles which is one of our scenarios. The agent's utility function gets higher when the agent pleases the user. So personalization is a key concept to get a higher utility.

In this paper, we consider the previous uses of the situation calculus in this area. We show that the situation calculus is a good basis for our formalization but is not enough, and we show that the situation calculus extended with guarded action theories is suitable. We propose a use for modelling applications that interact with the user and propose some kind of personalization. We express two different scenarios of personalization using situation logic and explore the idea that an application providing adaptation is a formal extension of the same application without adaptation in the logical model.

Section 2 introduces two scenarios of personalized systems. Section 3 presents the situation calculus and some of its variants. Section 4 presents our use of the situation calculus for the modelling of applications and illustrates it with the proposed scenarios. Section 5 concludes.

II. Scenarios

We choose different kinds of personalized systems to build our scenarios: a recommender system and an adaptive educational game.

A. News aggregator

We consider a simplified scenario of a news aggregator. The system proposes news articles to the user. For each

978-1-4577-1676-8/11/$26.00

article, the user states whether he read it and liked it, or ignored it. The system then uses this information to recommend the user articles he may like. This recommendation can be based on various factors. For example, the system might recommend articles that are related to articles the user liked.

This scenario can be extended to a full fledged application where news articles sources are handled, recommended, banned and so on.

B. Educative game for children

We consider a simplified version of a game to teach vocabulary to children. A picture and four words are presented to a child. He as to choose the word corresponding to the picture. Pictures of the same concept are presented again until the concept is known by the child. The concepts presented to the child are selected randomly among those unknown to him. The difficulty is adapted depending on the child's results. The use case is as follows:

- A picture and four words are presented to the child.
  - If the child selects the wrong word: the error is shown along with an image of the selected word.
  - If the child selects the right word: a congratulation message is displayed, and the success is remembered in order to know that the child knows the word.
- The system asks another question (a picture and four words).

III. Previous uses of the situation calculus

A. The choice of the situation calculus

We want to model the behaviour of a web application interacting with one or many users. For that, we need a formalism able to express the behaviour of the system from the point of view of the application, so that we can use the model to implement the application. We would also like a formalism that allows to prove the validity of our model.

We choose the situation calculus. This formalism models agents interacting with their environment and acting upon it. It has been used to express agents interacting on the web [3], [4] or to express a hypertext system [5]. It has also been used previously to express the personalization of adaptive hypermedia [6]. We want to explore its use to express the personalization in other types of personalized applications. Because the model is defined logically, it allows to prove all kind of properties on the modelled applications. For example, it enables the proof that there will always be a response from the application, whatever the user does.

An alternative logical framework is modal logic. Its use to model personalization has been studied in [7]. However, modal logic is much less suitable to actually program agents than situation calculus is. Modal logic has nothing similar to what golog is to situation calculus, hence our choice.

B. Original formalism

1) Situation calculus: The situation calculus [8] is a first order language designed to represent changing worlds. All changes are the result of primitive *actions*. Actions may take parameters. For example, $Drop(x)$ is the action of dropping an object x. We denote the *initial situation*, in which no action has happened yet, by s_0. All changes are the results of applying a sequence of primitive actions to the initial situation. The result of the application of an action α to a *situation* s is the situation $do(\alpha, s)$. Both actions and situations are first order terms.

Predicates whose values depend on the situation are called *relational fluents* (example: $Broken(x, s)$ denotes that the object x is broken in the situation s). Functions whose denotations depend on the situation are called *functional fluents* (example: $Position(x, s)$ is the position of an object x in the situation s).

In order to define a specific theory of actions, one needs to define a few axioms:

- For each primitive action $a(\vec{x})$[1], a precondition axiom of the form
$$Poss(a(\vec{x}), s) \equiv \pi_a(\vec{x}, s) \quad (1)$$
where $\pi_a(\vec{x}, s)$ is a formula specifying the preconditions of an action. For example:
$$Poss(Drop(x), s) \equiv Holding(x, s) \quad (2)$$
- For each relational fluent F , a successor state axiom of the form
$$Poss(a, s) \implies [F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s))] \quad (3)$$
where $\gamma_F^+(\vec{x}, a, s)$ (resp. $\gamma_F^-(\vec{x}, a, s)$) denotes whether the action changes the value of the fluent to true (resp. false). This axiom ensures that fluents that are not affected by an action do not change value during that action. Successor state axioms are defined in the same way for functional fluents.
- Axioms defining the initial situation.
- Axioms ensuring that different action names denote different actions.

2) ConGolog: ConGolog [9] (extension of golog [10]) is a high level logic programming language in which actions from the situation calculus are instructions. It enables the construction of complex programs by assembling the primitive actions of the situation calculus with various constructs. This language has been used to

[1] $a(\vec{x})$ is a shortcut for $a(x_1, \ldots, x_n)$

program robots [9] or agents of the semantic web [3]. The following constructs are available :

- a — primitive actions: a situation calculus action term.
- $\phi?$ — tests: only branches of a non-deterministic choice in which a ϕ is true can be executed.
- $\delta_1; \delta_2$ — sequences: δ_1 is executed, then δ_2 is executed.
- $\delta_1 | \delta_2$ — non-deterministic choice of actions : either δ_1 or δ_2 is executed.
- $\pi(x)\delta(x)$ — non-deterministic choice of parameters : $\delta(x)$ is executed for some value of x, where x is a free variable in $\delta(x)$.
- δ^* — non-deterministic iteration: δ is executed zero or more times.
- **if** ϕ **then** δ_1 **else** δ_2 — conditionals: if ϕ is true, δ_1 is executed, else δ_2 is.
- **while** ϕ **do** δ — while loops: δ is executed while ϕ is true.
- **proc** $P(\vec{v})\delta$ **endProc** — procedure: defines a procedure P that may be called later in the program.
- $(\delta_1 || \delta_2)$ — concurrent execution: δ_1 and δ_2 are executed concurrently. One of them may be blocked when it reaches a primitive action whose preconditions are false or a test whose condition is false. In this case, the program execution continues with the other process until the blocked process can resume execution.
- $(\delta_1 \rangle\rangle \delta_2)$ — concurrency with different priorities: same as the previous, except δ_2 can only execute when δ_1 is done or blocked.
- $\delta^{||}$ — concurrent iteration: like δ^*, but the instances of δ are executed in concurrency rather than in sequence.
- $\langle \phi \rightarrow \delta \rangle$ — interrupt: when the trigger condition ϕ becomes true, the body δ is executed, suspending any lower priority process.

C. Problem of the situation calculus with personalization

There is a problem if we want to model personalized applications as agents in the situation calculus, because they interact with users whose reactions are unknown. For example, suppose we want to present a news article to the user and ask him whether he liked it. There is no way to do that in basic situation calculus from the standpoint of the agent. In this configuration, only the agent's actions can change the world, and all these actions have predictable results. Asking the user if he liked something or not makes the world change in an unpredictable way.

A solution would be to define actions executed by the user. It is even done this way in [5]. But we would loose the clear cut between the agent (embodied into the application) and the environment it interacts with (the user). We also loose the benefit of modeling the agent itself: proving properties of its behaviour or building tools helping to design it or to enhance it.

To solve that problem, we use an extension of the situation calculus that allows us to consider the choices made by the user.

D. Some rejected approaches

We envisaged different modelling approaches to consider changes in the situation independent of the agent.

[11] introduced the theory of non-deterministic actions of the situation calculus. In this theory, actions can be defined with non-deterministic results. This theory wasn't developed later and is anterior to the guarded action theories that we use.

The sensing theory [12] is an extension of the situation calculus in which the knowledge of the agent is not complete. However, this formalism cannot be used to model a fluent that may change without direct control of the agent, for example a *like*. In the sensing theory, a fluent $Knows(\phi, s)$ states that the agent knows ϕ in a situation s and knowledge of the value of a fluent is acquired through an action modifying the $Knows$ fluent called a *sensing action*. Even though the value of a fluent might be known or not, it still has a value that obeys to the successor state axioms, and cannot change without an action. The guarded actions theories we use are based on sensing, generalizing the concept of sensed fluents to allow them to be either deduced by regression or sensed.

Another approach that we considered is based on the notion of noisy effectors [13]. Noisy effectors are golog procedures that result in an imprecise application of an action. They are of the form

$$\textbf{noisy-action}(x) \equiv \pi(y).\textbf{action(x, y)} \quad .$$

While the idea is interesting, the noisy effectors aim to model action whose consequences are imprecise and unknown, and that article's main study is the mitigation of the imprecision.

The uncertain result of actions has also been treated from the probabilistic point of view [14]. In this article, actions are extended to a couple $\langle input, outcome \rangle$, where *input* is the decision taken by the agent and *outcome* the probabilistic result (an *input* may have many potential *outcome*). But in the formalism as described by the authors, we are still stuck with the fact that the only actions undertaken are the agent's actions. So even if the proposed formalism considerably opens up the basic situation calculus, it still doesn't mean our requirements in terms of architecture.

IV. The situation calculus to model applications

A. Guarded action theories

To fulfill the need to take user response into account, we use guarded action theories [15]. In a guarded action theory, there is a finite number of *sensing functions* (or *sensor functions*), which are unary functions whose argument is a situation that model a sensor available to the agent. A *sensor-fluent formula* is a formula that contains at most one situation argument. Sensor-fluent formula that mention no sensor functions (resp. no fluents) are called *fluent formula* (resp. *sensor formula*).

A guarded action theory is like a basic action theory where the successor axioms are replaced by two sets of axioms:

- Guarded successor state axioms, of the form
$$\alpha(\vec{x}, a, s) \implies [F(\vec{x}, do(a, s)) \equiv \gamma(\vec{x}, a, s)] \quad (4)$$
where α is a fluent formula, F is a relational fluent and γ is a fluent formula. These axioms enable regression, i.e. deriving the value of a fluent in a situation from the value in previous situations.
- Guarded sensed fluent axioms, of the form
$$\beta(\vec{x}, s) \implies [F(\vec{x}, s) \equiv \rho(\vec{x}, s)] \quad (5)$$
where β is a sensor-fluent formula, F is a relational fluent and ρ is a sensor formula. These axioms enable sensing the value of an axiom in a certain situation, by using *sensing functions*.

The guards α and β are used to specify when the regression and sensing formulas are usable. This enables us to have fluents that can be sensed in certain situation, and can be modified by actions in other situations.

IndiGolog [16], [17] is an extension of ConGolog in which primitive actions are actions of a guarded action theory. It also introduces a new language construct, the search operator $\Sigma(\delta)$ which is used to specify that lookahead should be performed over the program δ to ensure that non-deterministic choices are resolved in a way that guarantee its successful completion. Outside of the search block, non-deterministic choices are resolved in an arbitrary way. Consequently, the computation is reduced because no lookahead is performed outside of search blocks.

B. Architecture

In our framework, the application is composed of an agent in IndiGolog that interacts with users through an interface that abstracts the interaction by providing sensor functions to the agent.

Figure 1 depicts our architecture: on the theoretical level, the action theory defined using the situation calculus enables us to prove some properties of the systems. This action theory is implemented using IndiGolog on the second level. Our web application is expressed using this implementation and interacts with the user. A CASE tool helps with the design of the personalized system.

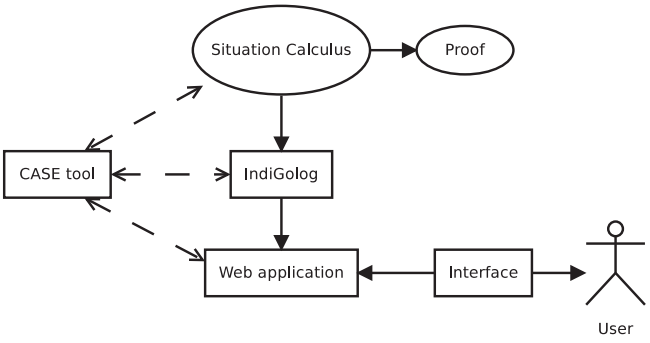

Figure 1. Architecture

C. Modeled scenarios

1) News aggregator: This application handles news articles. The relation $article(x)$ denotes that x is a news article. The relation $related(x_1, x_2)$ denotes that two articles are related to each other.

There are 3 fluents: $presented(x, s)$, $liked(x, s)$ and $ignored(x, s)$ which denotes that an article has been presented to, liked by or ignored by the user.

There is one primitive action: $present(x)$ (the agent presents an article to the user).

In the initial situation, nothing has been presented to the user:

$$\forall x.article(x) \implies \neg presented(x, s_0) \wedge \neg liked(x, s_0) \wedge \neg ignored(x, s_0) \quad (6)$$

We only present articles which have never been presented. The fluent *presented* is set to true by the action of presenting.

$$Poss(present(x), s) \equiv \neg presented(x, s) \quad (7)$$

$$True \implies [presented(x, do(a, s)) \equiv a = present(x) \vee presented(x, s)] \quad (8)$$

There are three sensor functions:

- **likedLastArticle**(s) tells if the user has liked an article by calling an external function.
- **passedLastArticle**(s) tells if the user has passed an article by calling an external function.
- **lastArticle**(s) returns the last presented article.

$$\textbf{lastArticle}(s) = x \implies liked(x, s) \equiv \textbf{likedLastArticle}(s) \quad (9)$$

$$\textbf{lastArticle}(s) = x \implies passed(x, s) \equiv \textbf{passedLastArticle}(s) \quad (10)$$

Also, we know that these fluents can only change when the article is proposed to the user. There is a similar axiom for the fluent $passed(x, s)$.

$$a \neq present(x) \implies liked(x, do(a, s)) \equiv liked(x, s) \quad (11)$$

We also introduce the fluent $recommended(x, s)$ that denotes an article is recommended for the user. A possible recommendation strategy could be to recommend articles that are related to articles that the user liked.

$$recommended(x, s) \equiv \exists x'.related(x, x') \wedge liked(x', s) \quad (12)$$

The behaviour of the application can then be expressed in IndiGolog:

$$(\Sigma(\pi(x).[recommended(x)?;\ present(x)]))^* \rangle\rangle (\pi(x).present(x))^* \quad (13)$$

The first process, which presents recommended articles to the user, has the highest priority. This ensures that only recommended items are presented to the user, as long as there are some. When there are no more recommended articles to present, the other process sends random articles to the user until a new item is recommended.

The first process uses the search operator to make sure only recommended articles are chosen in the nondeterministic choice. No search operator is used in the second process, so the article to present is chosen arbitrarily (possibly randomly).

It is possible to prove that this application will not stop presenting articles before all the articles are read.

2) Educative game for children: This application handles concepts that we want a user to learn. They are denoted by the relation $concept(x)$.

The fluents are:

- $proposed(x, s)$ — a concept is proposed to the user as an alternative in the current question.
- $rightanswer(x, s)$ — the concept is the right answer to the current question.
- $known(x, s)$ — the concept is known to the user.

The primitive actions are:

- $addWrongAnswer(x)$ — the application adds a wrong answer to the current question.
- $addRightAnswer(x)$ — the application adds a right answer to the current question.
- ask — the prepared question is asked to the user. This is only possible if a right answer has been added to the question.

There are also two sensor functions:

- **lastRightAnswer** returns the right answer to the last asked question.
- **correct** states whether the user answer correctly to the last question.

The preconditions axioms are:

$$Poss(ask, s) \equiv \exists x.rightanswer(x, s) \quad (14)$$

$$Poss(addWrongAnswer(x), s) \equiv \neg proposed(x, s) \wedge \neg known(x, s) \quad (15)$$

$$Poss(addRightAnswer(x), s) \equiv \neg proposed(x, s) \wedge \neg known(x, s) \wedge \neg(\exists x')rightanswer(x', s) \quad (16)$$

The guarded successor state axioms are:

$$True \implies [proposed(x, do(a, s)) \equiv a = addWrongAnswer(x) \vee a = addRightAnswer(x) \vee proposed(x, s) \wedge \neg a = ask] \quad (17)$$

$$True \implies [rightanswer(x, do(a, s)) \equiv \vee a = addRightAnswer(x) \vee rightanswer(x, s) \wedge \neg a = ask] \quad (18)$$

$$a \neq ask \vee \neg rightanswer(x) \implies [known(x, do(a, s)) \equiv known(x, s)] \quad (19)$$

The guarded sensed fluents axioms:

$$\mathbf{lastRightAnswer}(s) = x \implies [known(x, s) \equiv \mathbf{correct}(s)] \quad (20)$$

The golog program expressing the behaviour of the application is:

proc $addOneWrongAnswer()$
$\Sigma(\pi(x, addWrongAnswer(x)))$
endProc
proc $addOneRightAnswer()$
$\Sigma(\pi(x, addRightAnswer(x)))$
endProc
proc $prepareQuestion()$
$addOneWrongAnswer; addOneWrongAnswer;$
$addOneWrongAnswer; addOneRightAnswer$
endProc
$(prepareQuestion; ask)*$ (21)

The program defines three procedures:

- $addOneWrongAnswer$ adds a wrong answer by selecting one possible wrong answer.
- $addOneRightAnswer$ adds a right answer by selecting one possible right answer.
- $prepareQuestion$ adds three wrong answer and one right answer in order to prepare a question.

The main program consists only of: prepare a question, ask it and repeat.

D. Conclusion of the modelling

The successful modelling of our scenarios show that the chosen formalism is suitable for the modelling of web applications.

Some personalization could easily be added to both scenarios. For example, the equation 12 that defines the fluent $recommended(x, s)$ can be rewritten to take into account the user model. Let's introduce the fluent $interestedIn(y, s)$ that denotes the user is interested in the subject y, and the fluent $dealsWith(x, y, s)$ that denotes the article x deals with the subject y. We redefine $recommended(x, s)$ to be:

$$recommended(x, s) \equiv \exists x' \; related(x, x') \wedge liked(x', s) \\ \vee \exists y \; interestedIn(y, s) \wedge dealdsWith(x, y, s) \wedge x \neq x' \tag{22}$$

The same way, for the second scenario, we introduce a sensing function **lastAnswerTime(x)** that returns the time the user took to answer last time. Then, we use this value to update the user model with a fluent stating how many times the user took less than ten seconds to answer, and we use this value to change the difficulty. A faster user gets more difficult questions.

V. Conclusion

Situation calculus has been proposed as a formal frame for web applications for a long time. However, its precise use in this area has never been made very clear, especially in relation with personalized systems. We put back the formalism into the general AI theory of rational agents. We have shown that if we want to model personalization, we need one of the most advanced version of the situation calculus – namely guarded action theories – to model our agent. We illustrated its use by modelling two application scenarios.

We are now studying the modelling of the personalization by itself, in order to show that the difference between a personalized application and its non-personalized counterpart can be reduced to a syntactic modification.

References

[1] N. Stash, A. Cristea, and P. D. Bra, "Adaptation languages as vehicles of explicit intelligence in adaptive hypermedia," *International Journal of Continuing Engineering Education and Life Long Learning*, vol. 17, no. 4, p. 319–336, 2007.

[2] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice hall, 2010.

[3] S. McIlraith and T. Son, "Adapting golog for programming the semantic web," in *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, 2001, p. 195–202.

[4] ——, "Adapting golog for composition of semantic web services," in *Principles of knowledge representation and reasoning-international conference*, 2002, p. 482–496.

[5] R. Scherl, "A golog specification of a hypertext system," in *Logical Foundations for Cognitive Agents*. Springer-Verlag, 1999, p. 309–324.

[6] C. Jacquiot, Y. Bourda, F. Popineau, A. Delteil, and C. Reynaud, "Glam: A generic layered adaptation model for adaptive hypermedia systems," in *Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2006, p. 131–140.

[7] G. Antoniou, M. Baldoni, C. Baroglio, R. Baumgartner, F. Bry, T. Eiter, N. Henze, M. Herzog, W. May, V. Patti *et al.*, "Reasoning methods for personalization on the semantic web," *Annals of Mathematics, Computing & Teleinformatics*, vol. 2, no. 1, pp. 1–24, 2004.

[8] J. Carthy and P. Hayes, "Some philosophical problems from the standpoint or artificial intelligence," *Machine Intelligence*, 1969.

[9] G. D. Giacomo, Y. Lespérance, and H. Levesque, "Congolog, a concurrent programming language based on the situation calculus," *Artificial Intelligence*, vol. 121, no. 1-2, p. 109–169, 2000.

[10] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl, "Golog: A logic programming language for dynamic domains," *The Journal of Logic Programming*, vol. 31, no. 1-3, p. 59–83, 1997.

[11] C. Boutilier and N. Friedman, "Nondeterministic actions and the frame problem," in *Working Notes AAAI Spring Symposium 1995 — Extending Theories of Actions: Formal Theory and Practical Applications*, 1995, p. 39–44.

[12] R. Reiter, "On knowledge-based programming with sensing in the situation calculus," *ACM Transactions on Computational Logic (TOCL)*, vol. 2, no. 4, p. 433–457, 2001.

[13] F. Bacchus, J. Halpern, and H. Levesque, "Reasoning about noisy sensors and effectors in the situation calculus," *Arxiv preprint cs/9809013*, 1998.

[14] J. Baier and J. Pinto, "Planning under uncertainty as golog programs," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 15, no. 4, p. 383–405, 2003.

[15] G. D. Giacomo and H. Levesque, "Projection using regression and sensors," in *International joint conference on artificial intelligence*, vol. 16, 1999, p. 160–165.

[16] G. D. Giacomo, H. Levesque, and S. Sardina, "Incremental execution of guarded theories," *ACM Transactions on Computational Logic (TOCL)*, vol. 2, no. 4, p. 495–525, 2001.

[17] G. Giacomo, Y. Lespérance, H. Levesque, and S. Sardina, "Indigolog: A high-level programming language for embedded reasoning agents," *Multi-Agent Programming:*, p. 31–72, 2009.