2013 IEEE/WIC/ACM International Conferences on Web Intelligence (WI) and Intelligent Agent Technology (IAT)

Parametric reasoning agents extend the control over standard behaviors

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

Jean-Paul Sansonnet
LIMSI-CNRS
BP 133
F-91403 Orsay cedex France
Email: jps@limsi.fr

***Abstract*—In this paper, we study how to extend the control over the behavioral space of logically specified agents so as they can take into account parameters issuing from constraints and/or preferences that are external to their core reasoning process. Our approach is supported by the proposition of a set of generic transformations to apply to the original agent program that is expressed in IndiGolog, a variant of the Golog language. Several of these transformations exploit the non-determinism present in IndiGolog programs. We also propose an automatic process to apply all those transformations on the basis of a set of parameters. Three case-studies show the significance of the approach in situations where agents are in interaction with human users.**

I. Introduction

An agent-based model [1], [2] is a generic way of considering the interaction between a user and a computer system. Logically specified agents provide the extra advantage of logical reasoning, and the possibility to prove the soundness of their expression and behavior [3], chapters 2 and 7 of [4]. Such agents run an agent program specified in a dedicated language like Golog [5], DyLog [6], Placa [7], *etc.*

Since the beginning, research on logically specified agents has focused on the rationality and the genericity of the proposed behaviors. However, for some years now, there has been a fast development of applications and services where artificial agents are in interaction with the general public. Indeed, studies about agent/human interaction have shown that variability in the agent's behavior are required by the users [8]: subjects often criticize the mechanical behavior of traditional agents, especially during interactive sessions. Moreover the expression by an agent of human-like behaviors has a strong impact upon its believability [9] and its acceptability [10]. Writing a new agent for each variation of the behavior is obviously not an option. So there is a need for agents that adapt their behavior according to parameters issuing from the situation of interaction.

For long, designers have put aside those parameters: the human user with whom the agent is in interaction is a generic user without any specific preferences, and the agent program is focused on rationality, hence it does not have a personality. There are two challenges in handling those parameters. The first challenge is the resulting combinatorial explosion, and logic programs are very sensitive to this dimension. The second and possibly deeper challenge is to guarantee that all the various programs obtained by assigning different values to those parameters share a common behavior. This is the main goal of our proposal.

These considerations lead us to distinguish two main classes of agent behaviors:

- *Standard behaviors* are executed by traditional rational agents;
- *Parametric behaviors* are executed by agents adding to their rational process the handling of parameters associated with the situation of interaction.

We propose a constructive approach to the notion of a parametric reasoning agent[1], by extending a standard agent. The parametric behavior is derived from the standard behavior by applying transformations, hence extending the control over the standard behavior. We define a set of transformations that produce those variations of a given agent program while ensuring its overall functionality. We also define an automatic process that takes a standard agent program and an extension of its action theory provided by the application designer and results in a parametric agent program. We focus on two significant classes of situations where agents are in interaction with human users: *personalized* systems [13] and *personified* systems [14], [15]. We are interested in the action theory in which the agent program is executed. Hence, we will consider only features or personality traits that are relevant to change the way an action is undertaken. Both cases share the same principle that a profile is attached to the agent. In one case, it is the profile of the user and the agent uses this profile to satisfy the user; in the other case it is the profile the agent is expected to run with. We consider agents whose program is expressed with IndiGolog, a programming language for agents described in section II. The process itself and the definitions related to the profile are described in section III, and their use is demonstrated using three case-studies in section IV.

II. Agents

We first introduce the Golog languages family and we explain why we need one of the most advanced form of Golog, namely IndiGolog. Next, we detail two directions in which we expect to apply our program transformation technique :

[1] In the literature, the term 'parametric agent' refers also to several research topics, where it applies to the design process phase rather than the standard behavior of the agent. For example, in Parametric Design Agents (PDA), it is associated with reusable components for knowledge models [11] or intelligent parametric design tools supporting collaborative engineering [12].

978-1-4799-2902-3/13 $31.00 © 2013 IEEE
DOI 10.1109/WI-IAT.2013.105

personalization of an application interacting with a user and personification of a program agent.

A. The situation Calculus and Golog

The situation calculus [16] was designed to represent changing worlds as a set of formulas in first order logic. All changes are the result of primitive actions. There is a special constant s_0 that denotes the initial situation, and a function symbol do where $do(a, s)$ denotes the successor situation to s resulting from performing a. The state of the world in a situation is described by a set of fluents – predicates that depends on the situation. To define the world of an agent, the programmer creates an action theory representing the agent's belief about the state of the environment and the preconditions and effects of the actions.

Golog [5] is a procedural programming language defined on top of the situation calculus. It provides a set of constructs to express complex procedural programs. The interpreter for the language uses a situation calculus action theory to reason and find a provably correct execution of the program.

A number of constructs are available to define complex programs for the agent.

- a — primitive actions: a situation calculus action term.
- $\phi?$ — tests: a test on ϕ in the middle of a sequence blocks if ϕ is false. This is used to block branches in case of a non-deterministic choice: only a branch where ϕ is true can proceed.
- $\delta_1; \delta_2$ — sequences: δ_1 is executed, then δ_2 is executed.
- $\delta_1 | \delta_2$ — non-deterministic choice of actions: successfully executed if either δ_1 or δ_2 is successfully executed.
- $\pi(x)\delta(x)$ — non-deterministic choice of parameters: successfully executed if $\delta(x)$ is successfully 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.

Plain Golog does not provide a way for the agent to take into account changes in the environment that do not come from his own actions. This is a problem for the expression of embedded agents, which need to update their internal representations according to changes in the environment. IndiGolog [17] is an extension to Golog that provides a solution to that problem by using sensing. It enables the user to define sensing actions which update the value of sensing fluents from an external interface during online execution. The way IndiGolog handles sensing creates a clear cut between the agent and the environment which is needed for embedded agents. Indigolog is a rich language that has been used for example to express the personalization of adaptive hypermedia [18] or other kinds of personalized systems [19].

B. Application domains

In this study, we consider two main domains involving interactions between agents and human users. Each domain defines a class of applications for parametric agents with a specific orientation: user model and agent model. Each of these models is described by a set of parameters which are human personality traits in our application domains.

We propose to start with a standard behavior: one which is known to work in the absence of any parameter. We then transform this standard program to obtain an altered program. The standard program contains non deterministic alternatives which, in the absence of any specific parameter, are resolved by random choice. The altered program makes use of parameters to resolve those alternatives by explicit choice. New choices may also be introduced with conditions. We guarantee that when no parameter is given, the altered program falls back on the standard program. However, agents with distinct profiles all behave according to their specific profile.

1) Personalization: A personalized system is a system that reflects some features of the user in a user model and applies this model to adapt various aspects of the system to the user. Adaptive Hypermedia [20] are an example of such system.

Most of the time, personalization is achieved by plugging (or in the worst case hardwiring) ad-hoc rules for handling predetermined situations. This is not bullet proof and in many situations it may lead to an incomplete agent and also to difficulties in the debugging process of the agents.

Moreover [21] showed that there is no formal method to build such an agent. The main reason is that there is no formal definition of personalization. Intuitively, the behavior of a personalized version of an agent program should follow more or less closely its non-personalized version. The problem is with the definition of "more or less closely". Actually, for δ a (non-personalized) program and ψ a profile related formula, every **if** $(\psi)\delta'$ **then** δ could be a personalized version of δ, without any further condition on δ'. And we certainly do not want that because the global behavior would be out of control.

We make use of parametric agents to implement a restricted and well defined class of personalized agent programs. We impose that the personalized behavior differ only on the result of non-deterministic choices, or by modifications of actions. We also make sure that an empty profile does not change the program.

2) Personification: According to [22] and more recently to [23], the decision making process of rational agent can be influenced by psychological features such as personality traits. These authors claim that it makes their behavior look more 'human-like' (notion of 'believable agents' as defined by Bates [9]), thus increasing their acceptability factor [10], which is considered useful in new applications where agents are more and more in interaction with humans.

Research on computational implementation of psychological profiles [24], has focused recently on psychological theories involving *trait* taxonomies, such as the Five Factor Model (FFM) [25] and its subsequent versions, enriched with *facets*, such as the NEO PI-R taxonomy [26]. Using the psychological profile of the agent as parameters would be a new way to implement these psychological agents, or even to turn a strictly rational agent into a rational-and-psychological agent.

Assume an *affective* robot [27], [28], provided with psychological features – say personality traits [25], such as lazy,

brutal, cautious *etc.* Assume the robot is given a goal to visit-a-house, composed of n rooms r_i. This goal exhibits possible flexibility in its achievement because the robot is not strictly compelled to visit all rooms.

Now suppose that a given agent is provided with the psychological trait cautious; then it may refuse to enter locked rooms r such that $locked(r)$, while another agent, provided with trait brutal, may break the door of a locked room. Both agents reach the goal: they both have visited the house their own way.

III. Transformations over IndiGolog programs

In this section, we describe an automatic alteration process for IndiGolog programs to transform a standard agent into a parametric agent. First, at III-A, we introduce a new IndiGolog operator that we need in the transformed programs. Then, at III-B and III-C, we introduce the profile and explain how to link profiles to the actions of the program through affinity. Next, we introduce basic transformations that slightly change the behavior of an IndiGolog program at sections III-D and III-E. Finally, we define the alteration process that uses affinity to apply the basic transformations at section III-F.

A. Changes Needed in IndiGolog

IndiGolog has a non-deterministic choice operator $|$, but it does not allow to specify *preferences* upon the execution of one branch or the other, which we need to express the altered agent. We introduce the nondeterministic choice with preference operator denoted by $\rangle$. $\delta_1 \rangle \delta_2$ means "try δ_1, and in case of failure try δ_2". Note that as stated in [17], the IndiGolog implementation tries nondeterministic branches left-to-right, so the implemented $|$ has the same behavior as our new $\rangle$ operator. However, it is of theoretical importance to clarify when the choice between alternatives is random, and when there is a preference between those alternatives. The formal semantics for the operator are given by formulas 1 and 2 in terms of IndiGolog transition semantics : $Trans(\delta, s, \delta', s')$ means that by executing one elementary step of the program δ in the situation s leads to the situation s' with the program δ' remaining to be executed ; $Final(\delta, s)$ means that the execution of δ is finished in s.

$Do(\delta, s, s')$ [17] means that there is an execution of program δ that starts in s and terminates in s'.

$$Final(\delta_1 \rangle \delta_2, s) \equiv Final(\delta_1, s) \vee \neg\exists s".Do(\delta_1, s, s") \wedge Final(\delta_2, s) \quad (1)$$

$$Trans(\delta_1 \rangle \delta_2, s, \delta', s') \equiv \exists s".Do(\delta_1, s, s") \wedge Trans(\delta_1, s, \delta', s') \vee \neg\exists s".Do(\delta_1, s, s") \wedge Trans(\delta_2, s, \delta', s') \quad (2)$$

B. Attributes and Profiles

First of all, the application designer must define what can alter the behavior, namely a set of attributes which are linked to actions, and are used to decide if an action must be favored or avoided. The nature of the attributes depends on the type of alteration that is done to the agent. The relation between attributes and actions is detailed in section III-C.

Definition 1. An attribute is an atom. The set of all attributes A is defined by the application designer.

The behavior of an agent is altered by providing the agent with a positive or negative view toward attributes. A profile encompasses all views of a given agent. For each attribute, the agent can be absolutely opposed, slightly opposed, neutral, slightly favorable or absolutely favorable. These are respectively denoted by the integer values -2, -1, 0, 1 and 2 [23]. We consider that values of 1 and -1 indicate preference and should not alter much the program, but values of 2 and -2 indicate requirement, and may lead to invalid programs if they cause the agent to veto a critical part of the program. The 0 value denotes ambivalence or neutrality. The application designer should keep this in mind when defining a profile.

Definition 2. A profile is a function from the set of all attributes A to $\{-2, -1, 0, 1, 2\}$.

In the case of a personalized agent, the attributes refer to things that may or may not please the user, and the profile tells what pleases and displeases a specific user. In the case of a personified agent, the attributes refer to psychological connotations of actions, that the personified agent wants to favor or avoid. Such profile can be viewed as a psychological profile of the agent since it tells what kind of behavior it likes or dislikes.

C. Attributes and affinity of an action

To automatically alter a program, we need a way to tell if an action (*resp* a subprogram) is preferred to another action (*resp* another subprogram) with regard to a given profile. This requires that actions carry extra information, outside of the situation calculus, that can be used to know whether they are desirable, neutral or disliked, with regard to a given profile. This is implemented by attaching to actions attributes, similar to those in the profile.

For example, in the case of an affective robot with only few possible traits, we may have $A = \{lazy, brutal, cautious\}$. A possible profile p is defined by $p(lazy) = 2$, $p(brutal) = -1$ and $p(cautious) = 0$

Definition 3. $attributes$ is a function from the set of actions to the set of sets of attributes. For a given action a, $attributes(a)$ is the set of attributes attached to a and it is provided by the application designer.

We join the actions and the profile to get the affinity of an action to a profile: a numeric value telling whether an action is suitable for a profile.

Definition 4. The affinity function aff_p maps actions to numbers and is defined by

$$aff_p(a) = \sum_{at \in attributes(a)} p(at)$$

Thus, an action a_1 is preferable to another action a_2 for a given profile p if $aff_p(a_1) > aff_p(a_2)$. To transform entire programs, we need to know the affinity of a program. Thus, we extend the domain of aff_p to all programs with an inductive definition. The execution of the program not being known at compile-time, we must do approximations, like assuming that

a non-deterministic choice is random and that the affinity is the mean of its components.

Definition 5. We extend the affinity function aff_p to programs by inductive construction on all the operators of IndiGolog. For parsimony, only some cases are shown.

$$\begin{aligned} aff_p(\phi?) &= 0 \\ aff_p(\delta_1;\delta_2) &= aff_p(\delta_1) + aff_p(\delta_2) \\ aff_p(\delta_1|\delta_2) &= \frac{1}{2}(aff_p(\delta_1) + aff_p(\delta_2)) \\ aff_p(\delta_1\rangle\delta_2) &= aff_p(\delta_1) \\ aff_p(\pi x.\delta) &= aff_p(\delta) \\ aff_p(\delta^*) &= aff_p(\delta) \end{aligned}$$

Actions with arguments (which model verbs with objects) must be defined differently, because the attributes can depend on the argument (the meaning of an action depends on the objects on which it is applied). For example, the same action read can be hard when reading a scholar book but can be easy when reading a newspaper.

Definition 6. The function $attributes_a$ takes an action with an argument $a(x)$ and results in a list of pairs $< predicate, attribute >$. For each of those pairs, $a(x)$ has the attribute when x satisfies the corresponding predicate. This function is provided by the application designer.

$$attributes_a(a(x)) = \{< \phi_1, att_1 >, \ldots, < \phi_n, att_n >\}$$

See section IV-A for an example.

D. Program transformation

Our program transformation process manipulates a set of basic transformations which are defined in this subsection and the next one. The formal way to apply them is studied in subsection III-F.

As said above, our goal is to alter the behavior of agents by changing how they make choices. This requires the program to be under-specified: the execution must not be fully constrained by the program. Instead, the program must present choice points so to have some leeway for the execution and let the agent make choices at runtime.

Choices in IndiGolog programs appear in the non-deterministic choices constructs $|$ and π. We introduce transformations altering the choice in both constructs, that can be applied in any program where those constructs appear.

First of all, it is possible to replace any non-deterministic choice of a program by a preferential choice. The first transformation T1 is just a mean to force an order of evaluation for the two programs.

Transformation 1 (T1). $\delta_1|\delta_2 \rightarrow \delta_i\rangle\delta_j \qquad i,j \in \{1,2\}, i \neq j$

The next transformation T2 changes the order of evaluation of the argument in the non-deterministic choice of argument so that arguments satisfying a predicate ϕ are evaluated first. This is done by duplicating the construct and altering the first one so that the only valid executions are those where $\phi(x)$ is true, falling back to the original program if no such x exists. The fallback part guarantees that if the original program can be executed, then so can the transformed program.

Transformation 2 (T2). $\pi x.\delta(x) \rightarrow \pi x.\phi(x)?;\delta(x)\rangle\pi x.\delta(x)$ Where ϕ a formula containing only fluents.

Transformations T1 and T2 guarantee that the transformed program is as successfully executable as the original program. However, it is desirable to restrict the execution, for example to prevent some behavior that is not suitable for some user. We introduce two new transformations T3 and T4, which are similar to the first two, but with the fallback part removed. Transformation T3 entirely removes the non-determinism. These transformations are to be applied with care, because they can lead to programs that cannot be executed.

Transformation 3 (T3). $\delta_1|\delta_2 \rightarrow \delta_i$ with $i \in \{1,2\}$

Transformation 4 (T4). $\pi x.\delta(x) \rightarrow \pi x.\phi(x)?;\delta(x)$

Theorem 1. Let δ' be the result of the tranformation T1 or T2 applied to a program δ. If δ has a possible execution, then δ' has one too.

A program tranformed with T1 and T2 has the exact same possibilities as the original program, only evaluated in a different order. Thus, if the original program has a possible execution (ie a solution to its problem), the transformed program will at least one possible execution and will solve the problem too. Transformations T3 and T4, on the other hand, remove possibilities, and may break a program (by removing crucial parts). Therefore, they must not be applied lightly. For that, the alteration process ensures to only use those two when the profile present extreme values (2 or -2), thus giving the responsibility to the application designer.

E. Actions transformation

Until now, we have considered that the program to be transformed is under-specified and has non-deterministic choices and that transformations can be used to direct choices. This approach helps ensuring that the transformed behavior does not diverge much from the standard behavior. However, it is possible to introduce new choices in a program, as long as the new options are only variations of what is already there.

Thus, we want to modify programs by replacing actions by other actions that have slightly different pre and post-conditions, such as opening a door versus breaking a door open, or the same definition but different effects outside of the situation calculus theory, such as displaying a sentence in a text in bold font or in italic font.

This requires the application designer to group similar actions in families. A family is a set of actions that express the same operation but executed in a different way. The exact meaning of "same operation" depends on the domain and on what the actions are. Like for the actions, it is the application designer's responsibility to define a family of actions, and to decide which actions should be considered as doing the same operation.

Definition 7. The predicate $family(F)$ is true if F is a family of actions. $family$ is defined by the application designer.

An example of family is described in section IV-C

Transformation 5 (T5). If $A = \{a_1, \ldots, a_n\}$ is a family of actions

$$\forall a_i \in A \quad a_i \rightarrow a_1|\ldots|a_n$$

Transformation T5 replaces a given action by a choice among all actions in its family. It can be used in conjunction with transformation T1 to reorder the actions and replace $|$ by $\rangle$, in order to favor some actions upon others, and with transformation T3 to remove some actions from this choice.

Since the actions of a family achieve the same operation, replacing one by another does not interfere with the overall functionality of the agent.

F. Alteration process

The alteration process of a program δ is made of three steps that are executed in order: transformation of actions which are part of a family, transformation of non-deterministic choices of programs, and transformation of non-deterministic choices of arguments. It uses the affinity of actions and programs to determine how to apply the transformations. Therefore, on top of the usual IndiGolog application definition, the application designer is required to give additional definitions that are used by the process, summarized in the following table.

Name	Description	Section
$attributes$	list of attributes for each action	III-C
$attributes_a$	for actions with arguments	III-C
$family$	family of actions	III-E
p	profile	III-B

1) Actions: The program is recursively scanned to find actions a_0 such that there exists a family F and $a_0 \in F$.

For each such action a_0, a_0 is replaced by a choice between the actions of the family with T5. Then actions with non-positive affinity are dropped with T3 (we only introduce actions with positive affinity), a_0 is dropped only if its affinity is negative (we keep the original as fallback, except if it non desirable), actions with equal affinity are grouped in a non-deterministic choice and the preferential choice of T1 is used to link the groups (ensuring that actions with the most affinity are considered first during the execution, and actions with the same affinity are equally considered), resulting in the pseudo-program (3).

$$[\ \rangle_{k=2..1}(|_{a_i \in F | aff(a_i)=k} a_i)\ ]\ (\langle\ a_0 \text{ if } aff(a_0) = 0) \quad (3)$$

With a standard profile ($\forall a\ p(a) = 0$), there are no actions with positive affinity, and the result is a_0, the same as in the original program.

2) Choices of programs: The program is recursively scanned to find non-deterministic choices of programs. For each program of the form $\delta_1|\delta_2$, starting by the innermost, affinity of δ_1 and δ_2 are compared, and the one with the most affinity is preferred to the other using transformation T1.

If the difference of affinity exceeds 2 then the option with less affinity is removed using transformation T3.

With a standard profile, $aff(\delta_1) = aff(\delta_2) = 0$, and the program remains unchanged.

3) Choices of arguments: The program is recursively scanned to find non-deterministic choices of arguments. For each program $\pi x.\delta(x)$, the following steps are taken:

1) $\delta(x)$ is scanned to find actions a_i that have x as an argument.
2) For each action $a_i(x)$, $attributes_a(a_i)$ is a list of predicates associated to attributes. We gather all predicates in P. For each predicate, the associated attribute might be regarded by the profile as positive, negative, or standard. Thus, we might want to introduce the predicate, its negation, or not introduce it. If there are n predicates, this means that there are 3^n possibilities to study.
3) For each candidate of the form $c_k = \{\phi_1, \phi_3, \neg\phi_4, \ldots\}$, we compute the affinity of δ by giving each action an affinity equal to:
$$aff_p(a(x)) = \sum_{<att,\phi>} \begin{cases} p(att) & \text{if } \phi \in c_k. \\ -p(att) & \text{if } \neg\phi \in c_k. \\ 0 & \text{otherwise.} \end{cases}$$
4) We take the candidate c_k that maximizes the affinity of δ, and apply transformation T2 to replace $\pi x.\delta(x)$ with $\pi x.\psi(x)?;\delta(x)$ where $\psi(x)$ is the conjunction of the predicates of c_k. This ensures that arguments that satisfy $\psi(x)$ are considered before the others in the choice. If the affinity for c_k is superior to 2, T4 is used instead to make the condition mandatory. In case of equality of affinity, the candidate with the least predicates is taken.

With a standard profile, all affinities are null and the empty candidate is favored, thus leaving the program unchanged.

These three steps are executed in this order for a given profile p, and provide an automatic alteration process for IndiGolog programs.

IV. Case-studies

In this section, we discuss three cases-studies, involving distinct scenarios of alteration:

1) Personalization of recommender systems: recommender systems interact heavily with users. A lot has been done in the area of finding a recommendation that best suits each user. Most systems rely on statistics and/or machine learning: these systems cannot provide an explanation why something is recommended. We are interested in a logical approach to this problem and an IndiGolog agent in this area would be able to provide such an explanation.
2) Personalization of an Educational Hypermedia System (EHS): in these systems, agents have to deal with very complex environments. Agents are supposed to teach to users who expect a human level interaction. This is a good example for writing complex strategies that may need to be altered to take into account different user profiles.
3) Personification of a robot agent: in this situation, the agent is not directly in interaction with a user, at least during the achievement of the goal. However, a user can observe robot's actions and interpret its behavior in psychological terms.

A. Personalization of Recommender Systems

This first case-study deals with a Web-based application that recommends news articles. News articles are attached to topics. The standard agent chooses random articles to recommend, independently of the topic.

Depending on the user profile, we want the transformed program to avoid (or to prefer) articles that deal with certain

topics like sports or economy. We hide the details of implementation and focus on the IndiGolog program that is altered. We assume the existence of domain predicates to mark articles and their topics, as well as an action to push the recommendation to the user. The standard program is:

while $continue$
  do $\pi a.(article(a)?; \text{recommend}(a))$

For the transformation to work, we must define the attributes of the action. In this case, this is simple: the action has attributes *sport* if the article deals with sports, and *economy* if the articles deals with economy. There are some repetition in this definition, but it could easily be generated from the list of topics instead of being entered manually.

$$attributes_a(recommend(a)) = \\ [< dealsWith(a, sport), sport >, \\ < dealsWith(a, economy), economy >]$$

We execute the transformation processes on this program using two different profiles p_1 and p_2. $p_1(sport) = -2$, $p_1(economy) = 0$, $p_2(sport) = 1$, $p_2(economy) = -1$. The resulting program for p_1 only recommends articles that do not deal with sports:

while $continue$
  do $\pi a.(\neg dealsWith(sport, a)?; article(a)?;$
  $\text{recommend}(a))$

The program for p_2 prefers articles that deals with sport and tries to avoid articles dealing with economy, but still selects them if nothing else is available:

while $continue$
  do $\pi a.(dealsWith(sport, a) \wedge \neg dealsWith(economy, a)?;$
  $article(a)?; \text{recommend}(a))$
  $\rangle$
  $\pi a.(article(a)?; \text{recommend}(a))$

B. Personalization of an EH System

The second case-study deals with an EHS which contains pedagogical resources provided with a prerequisite relation. The user can access all the resources. In order to help the user find relevant resources, the altered version personalize the presentation by only providing relevant resources, *i.e.* those who are needed for the user to reach a given pedagogical goal.

There are two domain predicates: $resource(r)$ which means r is a resource, and $prereq(r, r')$ which means that r is a prerequisite of r'. We also use $prereq^*(r, r')$ which is the transitive closure of $prereq$. We assume a subprogram for displaying text on the student's screen. The standard program displays every available resource. The * operator executes its body as many times as possible.

$$(\pi r.resource(r)?; \text{display}(r))^*$$

To execute the transformation, we need to provide information about the attributes. We state that $display(r)$ has for attributes $goal(r')$, if it advances toward the goal r_i for each resource r_i, *i.e.* if r_i is a prerequisite of r_i.

$$attributes_a(display(r)) = \\ [< prereq^*(r, r_1), goal(r_1) >, \\ \ldots, < prereq^*(r, r_n), goal(r_n) >]$$

We can then transform the program using any profile stating a goal. For example, a profile where the goal is to be able to read and understand an article named "Mastering databases": $p(goal(masteringDatabases)) = 2$. It is set to 2, because we do not want to see anything that does not help for our goal. The resulting program with p only show articles useful for our goal:

$$(\pi r.prereq^*(r, masteringDatabases)?; \\ resource(r)?; \text{display}(r))^*$$

C. Personification of Robot Agent

The third case-study deals with a robot agent that circulates inside a building. There are several rooms, all closed and locked, containing keys to other rooms, or gold. The robot has an inventory of infinite size, containing things it caries. It can perform only two actions:

1) open a door d by using a key in its inventory: openWithKey(d);
2) once a door is open, the agent can take any item i in the room: take(i) .

The goal of the agent is to find and pick up the gold.

The program of the agent makes use of a procedure loot(r), to take all the items in a room. The main program loops over the closed doors and opens them until the gold is collected.

proc $\text{loot}(r)$
  while $\exists i.position(i) = r;$
    do $\pi i.(position(i) = r)?; \text{take}(i)$
endProc
proc main
  while $\neg position(gold) = inventory;$
    do $\Sigma(\pi r.closed(r)?; \text{openWithKey}(r); \text{loot}(r))$
endProc

In the initial situation, all doors are closed, room 1 contains the key to room 3, room 2 contains the gold, room 3 contains the key to room 2, and the agent has the key to room 1 in his inventory. A possible execution of this program is: openWithKey(1), take(key(3)), openWithKey(3), take(key(2)), openWithKey(2), take(gold).

We want to alter this agent by attaching to it personality features, as defined in section II-B. For that, we must add some flexibility to the agent to let it make meaningful choices. We introduce two new actions, that are different ways to open door: by breaking them (*break(d)*), or by picking the lock using a pick (*lockpick(d)*) that must be found first.

We attach to the agent a profile containing three traits: outlaw, brutal and practical. Technically, in FFM/NEO PI-R taxonomy [26], the trait outlaw is located in facet Dutifulness of Trait Conscientiousness, which covers the notions of: *loyal*-ness, *virtuous*-ness, *lawabiding*-ness and *truthful*-ness. brutal is associated with three facets: the negative pole of facet Tender-mindedness of Trait Agreeableness; and the facets Angry-Hostility, Impulsiveness of trait Neuroticism. The same method can be applied to a large subset of FFM/NEO PI-R.

To use the transformation process, we need to define that openWithKey, break and lockpick form a family (they all are ways to open doors). We also need to give attributes to

actions, which are in fact the psychological traits that favor these actions. Break gets the attribute brutal, lockpick gets the attribute outlaw, and openWithKey gets the attribute practical.

The attributes and families being defined, we can see how the alteration process affects the program, and the execution. Firstly, let's try a profile p_1 of an agent that is outlaw, dislikes brutality, and is not very practical: $p_1(brutal) = -1$, $p_1(outlaw) = 1$, $p_1(practical) = 0$. The program is modified to favor picking lock over using keys, but never breaks doors:

proc $\text{main} - p_1$
  while $\neg position(gold) = inventory$;
    do $\Sigma(\pi r.closed(r)?;$
      $\text{lockpick}(r)\rangle\text{openWithKey}(r);$
      $\text{loot}(r))$
endProc

Here is an example of execution in the same settings as before (with a lockpick added in room 3): openWithKey(1), take(key(3)), openWithKey(3), take(key(2)), take(pick), lockpick(2), take(gold).

Then, let's try a profile p_2 of a violent agent that is normally outlaw, but is very practical: $p_2(brutal) = 1$, $p_2(outlaw) = 0$, $p_2(practical) = 2$. In the resulting program, there is the ability to break doors, but a preference for opening doors with keys if possible.

proc $\text{main} - p_2$
  while $\neg position(gold) = inventory$;
    do $\Sigma(\pi r.closed(r)?;$
      $\text{openWithKey}(r)\rangle\text{break}(r);$
      $\text{loot}(r))$
endProc

An execution is: openWithKey(1), take(key(3)), break(2), take(gold).

In all three versions, the agent reaches its goal, through different means. We transformed the program according to a profile to give the agent a preference on the means to employ. Each altered agent solves the problem using its preferred actions if possible, falling back to others when needed, thus ensuring that any altered agent is able to solve the problem in some way.

V. Related Work

A. Different approaches to personalization

Other works have already used logically defined rational agents to perform personalization and adaptation tasks. [6] uses DyLog, a modal logic programming language, to create a personalized agent for adaptive tutoring. Their user profile (in this case, the learning goal) is a fluent like any other. Our approach is different because we set the user profile as parameters of execution apart from the domain fluents. Furthermore, their approach is an ad-hoc implementation of the personalized agent: there is no notion of standard agent from which the agent is evolved, so there is no way to guarantee the personalization does not interfere with the standard behavior of the agent.

In [29], the authors present a way to practically implement BDI-style agents on top of IndiGolog. The IndiGolog BDI agent program δ_P is obtained by inductively transforming the original BDI plan-body P program. This transformation makes use of two procedures named $achieve_e(\mathbf{x})$ and $handle(a)$. Those procedures rely heavily on non-determinism to choose an option in the plan library for a given event, and to call the appropriate procedure to resolve an event. Our approach is fully compatible with this mechanism. Surely, the original BDI agent could be parameterized by introducing new plans and rules to handle preferences in an ad-hoc way. But what we offer here is far cleaner and more powerful than that. We can apply our transformations to the overall obtained δ_P program, and especially to the $achieve_e(\mathbf{x})$ and $handle(a)$ procedures. This way, we can stick to the agent standard behavior while allowing variations around this behavior.

In [30], the authors extend IndiGolog to handle prioritized goals. They emphasize the need for the agent language to handle both declarative and procedural goals, because some goals should be declaratively stated as preferred to some others. This has been investigated further in [31] and [32]. This approach has some common points with ours: it introduces a way to specify that some goal should be preferred to another that is close to our "$\rangle$" operator. However, it focuses on explicitly declared goals while we focus on behaviors and tendencies based on arbitrary profiles. Furthermore, its goals influence the global execution of a program while our transformations change local choices.

B. Adding psychological features to BDI agents

Since works of Rousseau and Hayes-Roth [33], extensive research has been undertaken, especially recently, for implementing cognitive and/or psychological features in artificial agents. While [34] have implemented a psychological model, dedicated to emotions, based on traditional SOAR architecture, most authors have proposed improvements of BDI architectures. For example, using the BDI platform JACK, CoJACK [35] provides additional layers which intend to simulate *physiological* human capacities like the duration taken for cognition, working memory limitations (*e.g.* "losing a belief" if the activation is low or "forgetting the next step" of a procedure), fuzzy retrieval of beliefs, limited attention or the use of moderators to alter cognition.

However our approach is distinct from most studies using BDI because we focus on psychological traits rather than cognitive capacities. Moreover psychological parameters influence actions and plans rather than goals. These two remarks also apply to works close to our study [36] or [37], based on the architecture of conversational agent GRETA [38].

VI. Conclusion

We have introduced a first definition of a parametric agent and of an associated process to transform a non-parametric agent into a parametric agent. To do so, we exploit the non-determinism of IndiGolog programs. We also ensure that the parametric behavior is based on the original behavior. We have shown that this transformation is sound.

To the best of our knowledge, our approach is the first to propose an alteration process to transform a plain rational

agent into a parametric agent. This transformation has immediate applications in the domains of personalized agents and personified agents.

Our approach differs from other approaches of personalization where focus is always put on goals. Here we propose to parameterize behavior, which is different: our parametric agent is not goal driven, it is behavior driven.

Still, future work is needed in several directions. First, we want to investigate how to best characterize the different paths an agent can follow. Secondly, we want to study if it is sound to open up for more paths and under which conditions. Lastly, we want to reduce the amount of information the application designer has to provide, in particular by studying if a formal definition can be given for action families.

REFERENCES

[1] M. Wooldridge, "Agent-based computing," *Interoperable Communication Networks*, vol. 1, pp. 71–97, 1997.

[2] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 117, no. 2, pp. 277 – 296, 2000.

[3] M. Wooldridge, *Reasoning about Rational Agents*. MIT Press, 2000.

[4] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

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

[6] M. Baldoni, C. Baroglio, and V. Patti, "Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions," *Artificial Intelligence Review*, vol. 22, no. 1, pp. 3–39, 2004.

[7] S. Thomas, "The PLACA agent programming language," in *Intelligent Agents*, ser. Lecture Notes in Computer Science, M. Wooldridge and N. Jennings, Eds. Springer Berlin Heidelberg, 1995, vol. 890, pp. 355–370.

[8] M. Xuetao, F. Bouchet, and J. P. Sansonnet, "Impact of agent's answers variability on its believability and human-likeness and consequent chatbot improvements," in *Proc. of AISB 2009*, Edinburgh, 2009, pp. 31–36.

[9] J. Bates, "The role of emotion in believable agents," *Commun. ACM*, vol. 37, no. 7, pp. 122–125, 1994.

[10] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[11] E. Motta, *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1999.

[12] D. Kuokka and B. Livezey, "A collaborative parametric design agent," in *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, ser. AAAI '94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 387–393.

[13] G. Adomavicius and A. Tuzhilin, "Personalization technologies: a process-oriented perspective," *Commun. ACM*, vol. 48, no. 10, pp. 83–90, oct 2005.

[14] D. Rousseau, "Personality in computer characters," in *AAAI Technical Report WS-96-03*. AAAI Press, 1996, pp. 38–43.

[15] C. Castelfranchi, F. D. Rosis, R. Falcone, and S. Pizzutilo, "Personality traits and social attitudes in multiagent cooperation," *Applied Artificial Intelligence*, vol. 12, no. 7-8, pp. 649–675, 1998.

[16] J. McCarthy and P. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University, 1968.

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

[18] 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, pp. 131–140.

[19] G. Dubus, F. Popineau, and Y. Bourda, "Situation calculus and personalized web systems," in *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*. IEEE, 2011, pp. 569–574.

[20] P. Brusilovsky, "Methods and techniques of adaptive hypermedia," *User modeling and user-adapted interaction*, vol. 6, no. 2, pp. 87–129, 1996.

[21] G. Dubus, F. Popineau, and Y. Bourda, "A formal approach to personalization," in *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*. IEEE, 2011, pp. 233–238.

[22] B. Hayes-Roth, "What makes characters seem life-like?" in *Life-like Characters. Tools, Affective Functions and Applications*. Heidelberg: Springer, 2004.

[23] F. Bouchet and J. Sansonnet, "Influence of personality traits on the rational process of cognitive agents," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, vol. 2. IEEE, 2011, pp. 81–88.

[24] O. P. John, R. W. Robins, and L. A. Pervin, Eds., *Handbook of Personality: Theory and Research*, 3rd ed. The Guilford Press, 2008.

[25] L. R. Goldberg, "An alternative description of personality: The Big-Five factor structure," *Journal of Personality and Social Psychology*, vol. 59, pp. 1216–1229, 1990.

[26] P. T. Costa and R. R. McCrae, *The NEO PI-R professional manual*. Odessa, FL: Psychological Assessment Resources, 1992.

[27] E. Sisbot, L. Marin, and R. Alami, "Spatial reasoning for human robot interaction," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2281–2287.

[28] R. Gockley, R. Simmons, and J. Forlizzi, "Modeling affect in socially interactive robots," in *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on Robot and Human Interactive Communication*, 2006, pp. 558–563.

[29] S. Sardina and Y. Lespérance, "Golog speaks the BDI language," in *Programming Multi-Agent Systems*, ser. Lecture Notes in Computer Science, L. Braubach, J.-P. Briot, and J. Thangarajah, Eds. Springer Berlin / Heidelberg, 2010, vol. 5919, pp. 82–99, 10.1007/978-3-642-14843-9_6.

[30] S. Sardiña and S. Shapiro, "Rational action in agent programs with prioritized goals," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, ser. AAMAS '03. New York, NY, USA: ACM, 2003, pp. 417–424.

[31] C. Fritz and S. McIlraith, "Decision-theoretic Golog with qualitative preferences," in *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*, Lake District, UK, June 2006, pp. 153–163.

[32] S. M. Khan and Y. Lespérance, "A logical framework for prioritized goal change," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, ser. AAMAS '10. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 283–290.

[33] D. Rousseau and B. Hayes-Roth, "Personality in synthetic characters," in *Technical report, KSL 96-21*. Knowledge Systems Laboratory, Stanford University, 1996.

[34] J. Gratch and S. Marsella, "A domain-independent framework for modeling emotion," *Journal of Cognitive Systems Research*, vol. 5, no. 4, pp. 269–306, 2004.

[35] R. Evertsz, F. E. Ritter, P. Busetta, and M. Pedrotti, "Realistic behaviour variation in a BDI-based cognitive architecture," in *Proc. of SimTecT'08*, Melbourne, Australia, 2008.

[36] L. Malatesta, G. Caridakis, A. Raouzaiou, and K. Karpouzis, "Agent personality traits in virtual environments based on appraisal theory predictions," in *Artificial and Ambient Intelligence, Language, Speech and Gesture for Expressive Characters,achie AISB'07*, Newcastle, UK, 2007.

[37] M. McRorie, I. Sneddon, E. de Sevin, E. Bevacqua, and C. Pelachaud, "A model of personality and emotional traits," in *Intelligent Virtual Agents (IVA 2009)*, ser. LNAI, vol. 5773. Amsterdam, NL: Springer-Verlag, 2009, pp. 27–33.

[38] C. Pelachaud, "Some considerations about embodied agents," in *Int. Conf. on Autonomous Agents*, Barcelona, 2000.