Adding Preferences and Moral Values in an Agent-Based Simulation framework for High-Performance Computing (Full)

David Marin Gutierrez[3], Javier Vázquez-Salceda[1], Sergio Alvarez-Napagao[12], and Dmitry Gnatyshak[2]

[1] Universitat Politècnica de Catalunya-BarcelonaTECH, Edifici Omega. C/ Jordi Girona 1-3, E - 08034 Barcelona
{jvazquez@cs.upc.edu, salvarez@cs.upc.edu}
[2] Barcelona Supercomputer Center (BSC). C/ Jordi Girona 1-3, E - 08034 Barcelona
{dmitry.gnatyshak@bsc.es}
[3] PwC Spain. Av. Diagonal, 640, E - 08017 Barcelona
{maringutierrezdavid@gmail.com}

Abstract Agent-Based Simulation is a suitable approach used now-a-days to simulate and analyze complex societal environments and scenarios. Current Agent-Based Simulation frameworks either scale quite well in computation but implement very simple reasoning mechanisms, or employ complex reasoning systems at the expense of scalability. In this paper we present our work to extend an agent-based HPC platform, enabling goal-driven agents with HTN planning capabilities to scale and run parallelly. Our extension includes preferences over their objectives, preferences over their plans, actions, and moral values. We show the expresiveness of the extended platform with a sample scenario.

Keywords: agent-based simulation, goals, preferences, values

1 Introduction

Agent-Based Simulation (ABS) is a computational approach for simulating the activities and interactions of autonomous agents in order to better understand how a system behaves. Furthermore, they allow for the simulation of complex environments where perception, decision-making processes and actions carried out are dispersed among several stakeholders or agents. The purpose of ABS is therefore to obtain explanatory insight into the behavior of a group of agents which share a common environment. ABS can be applied to many fields such as biology, social sciences, ecology, economics, policy-making, etc. Specifically, ABS can be used to analyze the social relationships between agents by means of norms, moral values, and social conventions, their adherence to those norms and values, how they affect and limit their actions, and how they may change over time as the agents interact with each other and their environment.

Many ABS frameworks have been built focusing on large simulations to be run in High-Performance Computing (HPC) platforms. In current HPC-Based

ABS approaches [1] models may be elevated to and examined at genuinely large scales at the expense of having agents with limited interaction among them, perception and reasoning capabilities, sometimes even reducing agents to mere rule-based or functional input-to-output transformers. Some well-known ABS systems in this group include Repast[2], NETLOGO[3], and MASON[4]. An opposite approach are Multi-Agent frameworks (such as Jadex[5], 2APL[6], BDI4Jade[7] or GOAL[8]) that offer cognitive agents with more powerful practical reasoning capabilities, but at the expense of having very limited scalability. Many other approaches in literature offer different levels of reasoning and scalability[9].

In [10], Gnatyshak et al. present a custom Python-based BDI-agent simulation framework capable of both hosting agents imbued with complex reasoning capacity *and* running simulations with large numbers of these agents. Scalability is tackled in this framework by parallelizing via PyCOMPSs[11] the reasoning cycle of goal-oriented agents.

In this paper we address the issue of further enhancing this framework by giving agents the capability to deal with preferences over their objectives, preferences over the actions they take in order to accomplish those objectives and (moral) values, as a first step towards a powerful agent-based micro-simulation framework to analise the impact of social values, norms and conventions in large populations. One extra objective of this work is to explore how *far* we can go without using numbers to encode preferences. Generally, humans do not reason using hard numbers but in qualitative terms. However, all state-of-the-art approaches we have analised end up adding hard numbers to their selection strategy. So we will analise how *not* using numbers limits the expressiveness of our system, how severe this limitation is, and draw some conclusions as to whether it is acceptable to use numbers to attain a desirable level of complex reasoning.

This paper is structured as follows: in §2 we briefly describe the previous works we used as reference; in §3 we describe the conceptual model and how we added goals, preferences over goals, preferences over plans and actions, and support for the expression of moral values; in §4 we show how our additions to the model work in a sample scenario; and in §5 we conclude by discussing some limitations of the current model and extensions to be explored as future work.

2 Related Work

Our model of goals has been inspired by two agent frameworks with working implementations: GOAP and BDI4JADE.

GOAP [12] is the AI created for the enemies of the video game F.E.A.R, mainly formalized by Jeff Orkin. In GOAP, goals are represented by specifying a **desired state of the world** that agents strive to achieve. This desired state is described using the same structure used for the current state of the world, an agent's beliefs, actions' effects, etc. Agents can have many independent goals, but can only pursue one at the same time. In order to plan, an agent must have a set of available actions, a set of beliefs about the world and sensors to periodically update those beliefs, and a set of goals. Each goal has a current priority, and the agent will choose to plan for the goal with the highest current priority. GOAP uses numeric priorities (i.e., a quantitative relation rather than qualitative). A* is used to plan with a heuristic minimizing the weighted number of actions used to reach the desired state., i.e., minimize the sum of costs of the actions in the plan. We borrow such goals defined as desired world states (see §3.1).

Ingrid Nunes's BDI4JADE [7] platform provides a BDI layer on top of JADE [13]. It uses the same structure as Orkin's GOAP to represent goals (desired state of the world). It supports the declaration of different types of goals: 'belief goals' (goals that deal with states of the world described by boolean variables), 'beliefset value goals' (same as before, but variables are continuous or have more than two possible values), 'composite goals' (used to represent goals composed of subgoals which have to be achieved sequentially or in parallel), etc. It also differentiates between desires (non-committed goals) and intentions (committed goals). Plans are an ordered set of actions and are executed to achieve a specific goal. In BDI4JADE agents do not have a set of actions that they can use to build plans, but rather, they have a library of plans that the agents can choose from. Each plan in the library has some applicability conditions (equivalent to actions' preconditions) that are used in the plan selection process. We get inspiration from BDI4JADE on its plan selection strategy.

Our main inspiration for the modelling of preferences over goals comes from CP-nets[14]. Although our actual implementation is definitely not an implementation of a CP-net, the main inspirations we have drawn from them is to establish one default and many conditional preorder relationships over goals, and building a graph to both visualize them and interpret them. We also analised Dignum et al. approach in [15] to model values (to adapt it to model preferences over goals), but upon closer inspection, we decided not to follow thia approach since it uses numerical values and in this work we aim for a more qualitative approach.

In the case of preferences over plans, we drew a great deal of inspiration from Visser's work in [16]. It introduces the concepts of goals' properties, which we use extensively in our modeling of priorities over plans. We also make use of their mechanism for property propagation in our implementation. We should note that our implementation is simpler than theirs. For instance, the paper defines both properties of goals (discrete values that a property can take) and resources of goals (numerical values and intervals that represent how much of a resource -e.g., money, food- is being consumed by a goal or a sub-goal), but we chose to simplify the approach and add only properties of goals, as we want to explore a qualitative, scalar-free preference approach.

3 Conceptual model

A **multi-agent system** $\mathcal{M}$ is defined as the tuple $\mathcal{M} = \{E, \mathcal{A}^+, \mathcal{C}\}$ where E is an **environment**, in which the agents reside, that they can perceive, gather information from, and act on; $\mathcal{A}^+$ is a non-empty set of agents; $\mathcal{C}$ is a **controller**, defined as the tuple $\mathcal{C} = \{\mathcal{I}, inAcs\}$ where $\mathcal{I}$ is the inbox for all the agents' outgoing messages (supporting agent communication), and $inAcs$ is the set of all the actions to be exercised on the environment (regulating how agents access and act upon it).

An **agent** is defined as $\mathcal{A}_i = \{ID, msgQs, outAcs, Bh, \mathbb{B}, \mathbb{G}, g_c, \mathcal{P}_c, \mathcal{MP}, \mathbb{P}_g, \mathbb{P}_p\}$ where:

- $ID = \{AgID, AgDesc\}$ is $\mathcal{A}_i$'s identity data:
  - $AgID$ is the unique identifier of $\mathcal{A}_i$
  - $AgDesc$ is an arbitrary description of $\mathcal{A}_i$
- $msgQs = \{\mathcal{I}, \mathcal{O}\}$ is the set of $\mathcal{A}_i$'s message queues
  - $\mathcal{I} = \{\ldots, msg_i, \ldots\}$ is the Inbox, the set of messages sent *to* $\mathcal{A}_i$
  - $\mathcal{O} = \{\ldots, msg_i, \ldots\}$ is the Outbox, the set of messages sent *by* $\mathcal{A}_i$
  - $msg_i = \{AgID_s, AgID_r, performative, content, priority\}$ is a **message** sent from agent with $ID = AgID_s$ to the agent with $ID = AgID_r$, with the corresponding (FIPA-like) performative type, content, and priority.
- $outAcs$ is the set of **external actions** to be executed on the environment. It is composed of tuples of the form: $\{senderID, a^e\}$, where ID is the sender's ID, and a^e is the action that is being sent.
- $Bh = \{\mathbb{RG}, \mathbb{P}\}$ is $\mathcal{A}_i$'s **role behavior**
  - $\mathbb{RG}$ is the set of **role goals** associated with the Bh which $\mathcal{A}_i$ is enacting
  - $\mathbb{P}$ is the set of plans $\mathcal{P}$ associated with the Bh
- $\mathbb{B}$ is the set of $\mathcal{A}_i$'s **beliefs**. It uses the same world state structure as E
- $\mathbb{G}$ is the set of $\mathcal{A}_i$'s **own goals** (see §3.1).
- $g_c \in (\mathbb{G} \cup \mathbb{RG})$ is the current **committed goal** (see §3.1).
- $\mathcal{P}_c = \{\ldots, ab_i, \ldots\}$ is $\mathcal{A}_i$'s current **plan**, which is an ordered set of action blocks. Each **action block** $ab_i = \{\ldots, a_{ij}, \ldots\}$ is an ordered set of actions (each a_{ij} is an action). There are three types of actions: **internal actions** (actions that are executed by the agent in order to change their beliefs), **external actions** (actions that are sent by the agent to the controller in order to be executed on the environment to alter it), **message actions** (actions that are used to generate messages intended to other agents)
- $\mathcal{MP}$ is the **metaplanner**, a library of plans for each goal (see §3.2).
- $\mathbb{P}_g$ is the set of **preferences over goals** (see §3.3).
- $\mathbb{P}_p$ is the set of **preferences over plans** (see §3.4).

Our conceptual model extends the one presented in [10]. Our extensions are described in the following sections.

3.1 Adding goal structure

We extend the conceptual model in [10] by providing a formal model for goals: *what* they are, *how* they are defined, and how they are *related with plans*. We have chosen to model goals as desired states of the world that agents strive to achieve. It is equivalent to the concept of **desires** in BDI. A goal is therefore defined by a collection of subsets of the variables that describe a state of the world (its **conditions**), and an assertion of their desired value(s). These conditions are expressions such as 'cash==10' or 'speed>=50' to mean that having exactly 10 units of cash and that maintaining a speed of 50 or above are part of the desired state of the world, respectively. Each subset describes a conjunction of variables that describe a desired state of the world and, in order for a goal to be considered achieved, it is required that all the variables of at least *one* of these subset have the desired values in the eyes of the agent (its **beliefs**).

We formally define the structure of a **set of goals** $\mathbb{G}$ as an unordered set of the form $\mathbb{G} = \{g_1, g_2, \ldots, g_n\}$ where each g_i is an individual goal among the many goals an agent has. A **goal** is defined as $g_i = \{name, descr, \mathbb{C}, status\}$ where *name* is a unique identifier of the goal, *descr* is an optional text describing the goal, $\mathbb{C}$ is the set of conditions over the state of the world for the goal to be considered achieved, and *status* is a boolean value that is *True* if and only if the conditions $\mathbb{C}$ are satisfied according to the agent's current beliefs $\mathbb{B}$.

A **set of conditions over the state of the world** is defined as unordered collections of assertions over the state of the world (the *environment*) of the form $\mathbb{C} = \{a_1, a_2, \ldots, a_n\}$ where $a_i = \{n_1 \star v_1, n_2 \star v_2, \ldots, n_m \star v_m\}$ is a conjunction of statements over the values of variables of the agent's beliefs, defined by n_i, which is the *unique* name of a variable of the agent's beliefs; $\star$, which is a binary operator ($\{=, \neq, >, \geq, <, \leq\}$); and v_i, which is the value of interest that is being asserted to n_i.

The agent possesses the capabilities to check whether or not an individual goal has been achieved according to its beliefs: $check\_goal(g_i, \mathbb{B})$ outputs *True* if, according to the agent's beliefs, the conditions of the goal have been met, and false otherwise. Our agents are allowed to have multiple goals, but are restricted to pursuing only one at a time. This *commitment* to a goal that is intended to be pursued (g_c in the agent tuple) is equivalent to the concept of **intention** in BDI. Agents have the capability to re-consider which goal they want to pursue, and may change the goal they are committed to even if they have not achieved it, depending on their current beliefs and the state of the world they perceive.

3.2 Adding a library of plans

We also extend [10] to enable specifying different plans for each goal, and to pick different plans for a committed goal with an element that will act as a library of plans, The implementation of the means-ends reasoner for the platform is a Hierarchical Task Network (HTN) planner[17]. The HTN is a tree composed of three types of nodes: (i) Primitive Tasks, (ii) Methods, and (iii) Compound Tasks. The root of the HTN is an abstract compound task (e.g., *order food*). Figure 1 provides an example. Our agents have a library of predefined HTN plans that the agent can pick from, and these plans will be related to goals by means of the structure of the **metaplanner**, which is the $\mathcal{MP}$ element of the agent tuple. Formally, it can be viewed as $\mathcal{MP} : \mathbb{G} \longrightarrow \mathbb{P}^*$, a matching relationship from goals towards plans, where $\mathbb{P}$ is the set of plans $\mathcal{P}$ associated with goal g_i and $\mathbb{P}^*$ is used to indicate that it can output tuples of plans of arbitrary cardinality (meaning one specific goal may have, for instance, three plans associated to it, while a different goal might have five, or two). We need also to add applicability conditions to plans: $\mathcal{P} = \{\mathbb{C}, ab_1, \ldots ab_n\}$, where $\mathbb{C}$ is the set of conditions over the state of the world (see §3.1) that determine a plan to be applicable, and each ab_i is an action block.

Other noteworthy aspects of the metaplanner are that it incorporates appropriate functions for plan selection. Therefore, it will not simply act as a library/collection of plans, but it will also perform part of the reasoning. This reasoning includes both checking which of the associated plans are available for application, as well as ordering them based on the preferences[4]. For the first functionality, the metaplanner features a $get\_available\_plans(g_i, \mathbb{B})$ function which, taking into account the current beliefs of the agent, it outputs a subset of the set of plans associated with the goal, containing only all plans that are applicable. For the second functionality, the metaplanner has a $pick\_plans(g_i, \mathbb{B}, prefs_{\mathcal{P}})$ function, where $prefs_{\mathcal{P}}$ are the agent's preferences over plans, that will pick the plan that is more adequate to the current situation according to the agent's preferences and beliefs, from among all the applicable plans.

3.3 Adding preferences over goals

The next extension we introduce in the model are preferences over goals. As we explained in §2 we drew inspiration from CP-nets and conditional preference formulas, to some extent, but we simplified the approach in order to be able to work without scalars, that is, having a fully qualitative approach to specifying preferences over goals.

To define preferences over a set of goals, the approach we have taken is to establish a strict partial order relation between them to indicate which goals must be pursued before trying to achieve other goals. These binary relations between goals are reflexive, transitive and assymetric. To model the context-dependent nature of preferences, we allow the declaration of conditional preferences, which are also a strict preorder relation over goals, but they only apply when their trigger conditions are met. A nice property of strict preorders is that they have always a unique direct acyclic graph (DAG) associated to them.

In order to encode **preferences over goals** in our agents, we have added the following element, $\mathbb{P}_g$ (which stands for "$\underline{\mathbb{P}}$references over $\underline{\text{g}}$oals") to the agent tuple. We define it as $\mathbb{P}_g = \{dGP, cGP_1, cGP_2, \ldots, cGP_n\}$, where dGP are the *default* preferences over goals (they apply under 'normal' circumstances), and cGP_i are *conditional* preferences over goals (they have some trigger set of conditions $\mathbb{C}_i$ over the state of the world as defined in §3.1).

The dGP and each cGP_i are defined as a DAG that corresponds directly to a **strict partial order** relationship between goals, and the only difference between them is that the dGP is the one active by default (does not need any conditions to be met), while the various cGP_i become active and replace dGP if some associated conditions are true.

Once all the strict preorder relations have been established, we deduce their associated DAGs. From those DAGs, we compute a valid topological ordering of

[4] We describe how we model preferences over plans in §3.4

each, and these orders are the ones in which goals will be pursued by the agents (by choosing the first non-achieved goal in the topological ordering), e.g.:

- We have one agent $\mathcal{A}_{\prime}$ which has the goals $\mathbb{G}_0 = \{g_0, g_1, g_2\}$. g_0 is a goal to tidy the agent's bedroom, g_1 is a goal to tidy the agent's kitchen, and g_2 is a goal to store clothes that are hanging out to dry in the open.
- If we denote "goal i must be achieved before goal j" as $g_i \rightarrow g_j$, the **default preferences** over goals of agent $\mathcal{A}_{\prime}$ are $\{g_0 \rightarrow g_2, g_1 \rightarrow g_2\}$, that is, before storing the clothes that are outside, $\mathcal{A}_{\prime}$ must have cleaned both his bedroom and his kitchen. Notice how both g_0 and g_1 must be accomplished before focusing on g_2, but there is no established order between g_0 and g_1, as it is a strict *partial* order. A valid topological ordering might be: g_0, g_1, g_2, but also g_1, g_0, g_2. By default, $\mathcal{A}_{\prime}$ will pursue his goals in either of those orders.
- The set of **conditional preferences** over goals of agent $\mathcal{A}_{\prime}$ is $\{g_2 \rightarrow g_0, g_1 \rightarrow g_0\}$ with the associated trigger conditions that the variable 'raining' must be $True$. If it is raining, the agent's top priority goal will be to collect the clothes, then cleaning their kitchen or bedroom, in no specific order. Therefore, the moment it starts to rain, $\mathcal{A}_{\prime}$ will switch to any of the topological orderings that can be given to this set (for instance, g_2, g_1, g_0)[5].

3.4 Adding preferences over plans and actions

By adding preferences over goals we provide agents with the capacity to choose *what* to pursue. But we also need to provide them with means to have preferences over *how* to achieve what they are pursuing. We humans have preferences not only over *what* goals we want to achieve, but also over *how* we want to achieve them, and these preferences may be context-dependent. Some people might prefer to drive to their workplace, while some others would rather walk there. But the preference on walking may change in the case the weather is very cold or rainy, then prefering to commute to work by a combination of transportation modes. In order to encode **preferences over plans and actions** in our agents, we have added element $\mathbb{P}_p$ (which stands for "$\underline{\mathbb{P}}$references over $\underline{\textbf{p}}$lans") to the agent tuple. We define it as $\mathbb{P}_p = \{gP_1, gP_2, \ldots, gP_n\}$.We denote the preferences over plans for each goal g_i by $gP_i = \{dPP, cPP_1, cPP_2, \ldots, cPP_n\}$, where dPP are the *default* preferences over plans for goal g_i (under 'normal' circumstances), and cPP_i are *conditional* preferences over plans for goal g_i (they have some trigger set of conditions $\mathbb{C}_i$ over the state of the world).

A **property of a goal** is the name of a variable of interest that a goal has the capacity to alter. Said variable does not necessarily have to be the name of a variable in the set of beliefs of an agent. It is simply something noteworthy that achieving a goal has the capacity to give a specific set of values. For example, if a goal is to 'order dinner', some of the properties might be 'vegetarian' and 'cuisine', and their possibles values might be $\{True, False\}$ and {'French', 'Italian', 'Spanish', 'Turkish'}, respectively. In our model each

[5] In case of conflicts between preferences, the default behaviour is to choose by order of declaration in the HTN. This can be overriden by the designer. Refer to §5.

goal, plan, subplan, and action may have a set of properties PS, of the form $PS = \{prop_1, prop_2, \ldots, prop_n\}$, and each property $prop_i$ is of the form $prop_i = \{v_1, v_2, \ldots, v_n\}$ where: $prop_i$ is the *unique* name/identifier of the property, and v_i is one of the possible values that the property can take. These values can be boolean, numeric, etc., depending on the nature of the property itself. The set of values that make up each property are used to indicate possible values the property can take. All properties can have the special $None$ value inside the set of their possible values. The presence of this value in a property of a plan or subplan indicates that said plan or subplan can be achieved through one or more actions that do not use or alter the property in question at all.

Propagation of properties consists in sending the properties 'upwards' from the most concrete actions, up to the root goal, passing through every subplan and subgoal in the way. The full description of the method is provided in [16]. Given two *sequential* actions that have the same parent, the parent's set of properties will be the result of computing the union between the two children's properties. Each child will not have different possible values for the same properties, since they are sequential actions, and it would not make sense to design a plan in which child action no. 1 sets 'cuisine'='Spanish' only for the child action no. 2 to set the cuisine to be 'French'. Therefore, the properties of the two (sequential) children will always be different, and the resulting properties of the parent node will simply be the joining of the children's sets of properties, and it is trivial to see that this process applies to n sequential children actions.

Given two *alternative* actions that have the same parent, the parent's set of properties will be the result of merging the properties of the children in the following manner: if both children set different values for the same property then, for the father, the values of the property will be the union of the values that the children had (e.g., if child no. 1 had 'cuisine'='Spanish' and child no. 2 had 'cuisine'='French', the parent task will have 'cuisine'={'Spanish', 'French'} to indicate that if that node is chosen, we will limit the possible values of 'cuisine' to those two values). If either child has a property that the other does not, the parent will simply take the same properties of the child that has it, and will add the special value $None$, to indicate that if that node is chosen, there is a path of the plan that accomplishes the goal without ever giving a value to that property.

Figure 1 provides an example of property propagation. It shows the set of plans associated to a goal of ordering dinner. There are three possible options: a plan to order burgers, a plan to order falafel, and a plan to order pizza. We assume that there is only a local burger, a local falafel, and both a local pizza restaurant and a big company that makes pizza. Other assumptions that we take are that all burgers and pizzas are non-vegan, and that all falafels are vegetarian. The designer only needs to declare properties on the actions. Then, as a result of the property propagation process, all vertices have their own set of properties that have propagated upwards, from the leaves (actions). Notice how, in general, all properties have propagated towards the upward nodes. However, most of these propagations have been very simple ones: from single child to parent, although there are two cases worth mentioning. The first one is the propagation from the

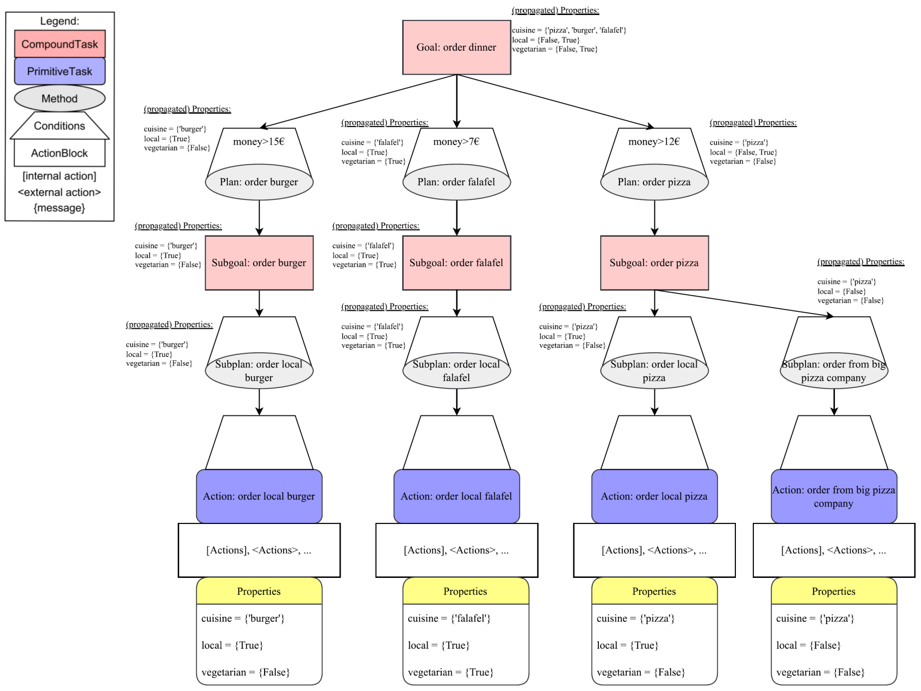


Figure 1. Property propagation on an HTN plan assocated to the *order dinner* goal

subplans to order local pizza and order from big pizza company. Notice how their properties are the same in all fields except for the 'local' field, with one holding it as True, and the other as False. However, these two *alternative* subplans share a common parent, and when their properties are propagated to it, they are merged in the way we described earlier: the parent has its property 'local' with *all* of its children values, to represent that, if that subgoal (or its parent subplan) is picked, then we can still order from either a local restaurant or a big chain. The other note-worthy example is the propagation of properties to the root node, where all options have been compiled in its properties.

3.5 Selection of plans and actions using properties

We will now briefly describe the process of choosing a plan taking preferences into account. An assumption we make throughout this whole example is that all plans are available, that is, our choices are not restricted by the environment in any way, shape, or form. Given a concrete goal g_i (order dinner) an agent has a set of preferences over the plans to achieve g_i. We can define this set as $gP_i = \{dPP, cPP_1, cPP_2\}$, where dPP is the default set of preferences, and cPP_1, cPP_2 are conditional sets of preferences. We assume that we have the following preferences over how to achieve the goal to order dinner (see Figure 1):

1. $dPP = \{cuisine = \{falafel\}\}$: by default the metaplanner would only follow the branch with this property, and order from the falafel restaurant.
2. $cPP_1 = \{cuisine = \{burger, pizza\}, local = \{True\}\}[weather = snowy]$: in case of snow the metaplanner would follow branches that are either falafel or pizza cuisine, but only those that are local (in the case of pizza this restricts it to only the local pizza place option).
3. $cPP_2 = \{local = \{False\}, vegetarian = \{False\}, cuisine = \{burger\}\}[weather = rainy]$: in case of rain the metaplanner attempts to follow branches meeting all the conditions, but even if the agent prefers to order non-vegetarian burgers, the first property prevails and leads to the only non-local option (pizza from big company).

As we can see, the agent picks from all the plans that satisfy the leftmost property, then, from those plans, it picks from those that satisfy the next leftmost property, etc. This process is for both default and conditionally triggered preferences, as they have the same structure, the only difference being that the latter need to be activated in order to take over and replace the default properties.

3.6 Adding values

Moral values can be simulated using the system of preferences over plans and actions described in §3.4. Consider the previous example of ordering food. We can ingrain moral values into each plan as extra properties. For instance, in our food ordering example (see Figure 1) primitive tasks are associated to a *local* value (meaning the social value to favour local businesses and products over globalization-oriented trade of products coming from far away). Another example is provided in Figure 2, where bike and walk options for transportations are positively associated to the *environmentalist* value.

As we are associating values to the primitive tasks, this may look as if our model pressuposes moral absolutism[6], but actuality, that is not true. As properties are defined for each plan of each agent, we can create an agent who thinks that lying is morally wrong, and an agent that thinks that it is morally right. Also, since the same action can be part of different subplans, we can also encode the fact that the morality of actions depends on their context. For example, if an agent kills an animal as part of a subplan to have fun, we can label that action as morally evil, but if the same agent kills an animal in his job as a veterinarian, then that action was not morally evil.

4 Example Scenario

We present a complex scenario to show how our agents fare with the extensions. The simulation consists of 64 steps. It starts at 08:00, and ends at 00:00 of the next day. Each simulation step corresponds to 15 minutes in the town. By default, the town starts with clear weather. Every iteration, there is a 10% chance of the

[6] Moral absolutism is the position that there are universal ethical standards that apply to actions, and according to these principles, these actions are intrinsically right or wrong, regardless of what any person thinks, or context.

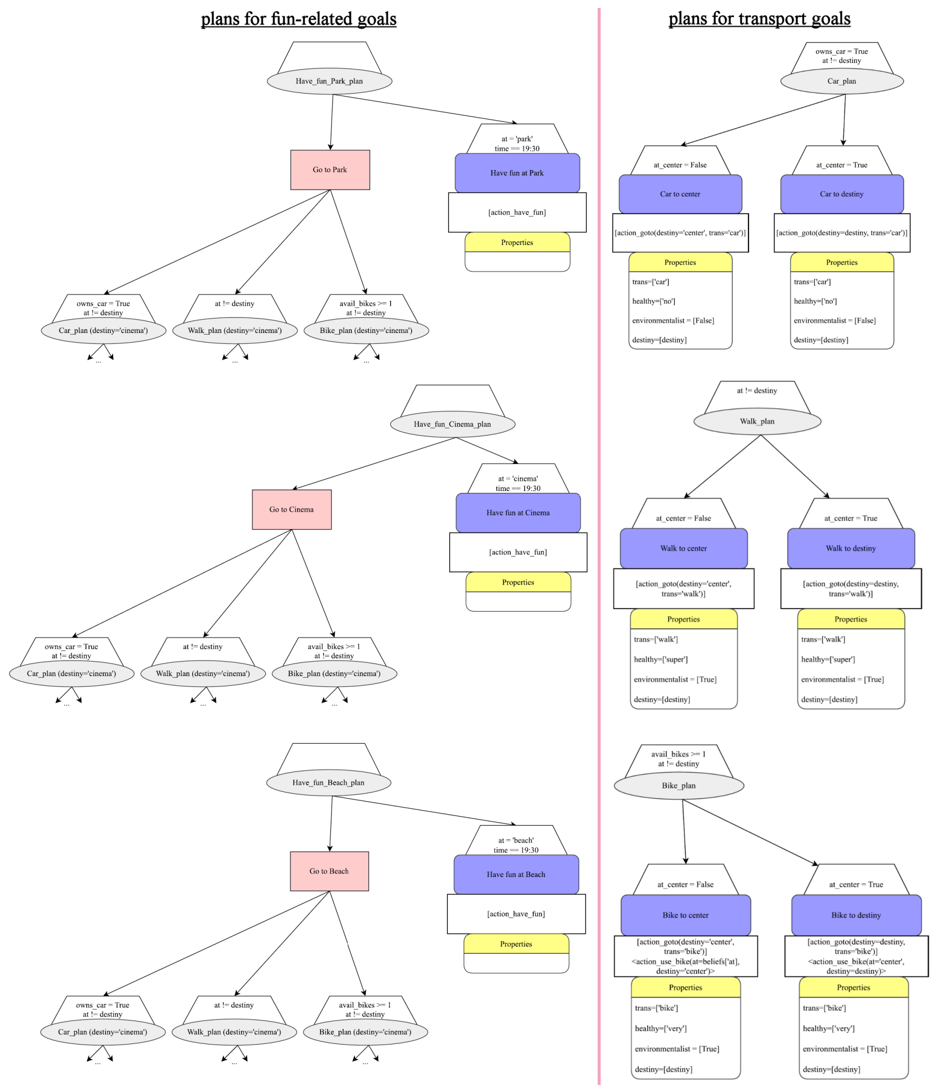


Figure 2. Library of plans for fun-related and transport goals

weather changing. If that chance happens, there is a 60% chance of the weather becoming clear, 30% chance of becoming cloudy, 9% chance of raining, and 1% chance of snowing. At every iteration, there is also a 0.2% chance, for every agent, to experience a medical emergency. All these parameters are configurable by the user. The environment is randomly generated using a *seed*, and the agents will react and plan accordingly to the changes on the environment. The agents' environment is a small town with some citizens living in it. These citizens are people which have their own set of daily goals (e.g., go to their workplace, have

Table 1. Alice's and Bob's goals, preferences and values.

ALICE's goals g_1 - Take children to school g_2 - Go to work g_3 - Work g_4 - Go collect her kids to school g_5 - Have fun with her kids g_6 - Go back home g_7 - Eat dinner g_8 - Attend any medical emergency	**BOB's goals** g_1 - Go to work g_2 - Work g_3 - Have fun g_4 - Go back home g_5 - Eat dinner g_6 - Attend any medical emergency
ALICE's preferences over goals Default: $g_1 \rightarrow g_2 \rightarrow g_3 \rightarrow g_4 \rightarrow g_5 \rightarrow g_6 \rightarrow g_7$ Conditional preferences: – if (**medical emergency**) $g_8 \rightarrow g_1 \rightarrow g_2 \rightarrow g_3 \rightarrow g_4 \rightarrow g_5 \rightarrow g_6 \rightarrow g_7$ – if (**snowing**) $g_2 \rightarrow g_3 \rightarrow g_4 \rightarrow g_5 \rightarrow g_6 \rightarrow g_7$	**BOB's preferences over goals** Default: $g_1 \rightarrow g_2 \rightarrow g_3 \rightarrow g_4 \rightarrow g_5$ Conditional preferences: if (**medical emergency**) $g_6 \rightarrow g_1 \rightarrow g_2 \rightarrow g_3 \rightarrow g_4 \rightarrow g_5$
ALICE's values – For **transport goals**: • environmentalist = $False$ – For **food-related goals**: • sustainability = $False$ • local = $False$ #big chains	**BOB's values** – For **transport goals**: • environmentalist = $True$ – For **food-related goals**: • sustainability = $True$ • local = $True$ #local businesses
ALICE's preferences over plans (transport goals) Default: $Car \rightarrow Walk \rightarrow Bike$	**BOB's preferences over plans (transport goals)** Default: $Bike \rightarrow Walk \rightarrow Car$ Conditional preferences: – if (**cloudy**) $Walk \rightarrow Bike \rightarrow Car$ – if (**rainy or snowy**) $Car \rightarrow Walk \rightarrow Bike$
ALICE's preferences over plans (fun-related goals) Default: $Fun\_in\_Beach \rightarrow Fun\_in\_Park \rightarrow Fun\_in\_Cinema$ Conditional preferences: – if (**cloudy**) $Fun\_in\_Park \rightarrow Fun\_in\_Cinema \rightarrow Fun\_in\_Beach$ – if (**rainy or snowy**) $Fun\_in\_Cinema \rightarrow Fun\_in\_Park \rightarrow Fun\_in\_Beach$	**BOB's preferences over plans (fun-related goals)** Default: $Fun\_in\_Beach \rightarrow Fun\_in\_Cinema$ Conditional preferences: – if (**cloudy or rainy or snowy**) $Fun\_in\_Cinema \rightarrow Fun\_in\_Beach$
ALICE's preferences over plans (food-related goals) Default: $Order\_Pizza \rightarrow Order\_Chinese \rightarrow Order\_Burger$ Conditional preferences: – if (**rainy**) $Order\_Chinese \rightarrow Order\_Pizza \rightarrow Order\_Burger$	**BOB's preferences over plans (food-related goals)** Default: $Order\_Pizza \rightarrow Order\_Burger \rightarrow Order\_Chinese \rightarrow Order\_Falafel$

fun, eat dinner, etc.). Like real people, they have preferences over *in which order* to pursue their goals, as well as preferences over *how* to achieve them. Finally, they might have some moral inquiries into the actions we perform (e.g., being environmentalists and thinking the usage of cars is immoral, etc.).

The `environment` class implements the map of city locations as well as other variables such as the current weather, the time, and extra internal variables for purposes of running the simulation. When an agent perceives the environment, they will only perceive the current time, the current weather, and the information of the location that they are currently in. For instance, if an agent is at the city center, it will not update its information about the state of the school, only about the state of the city center, the weather, and the time.

There are two main actors in our environment, **Alice** and **Bob**. They both are complex agents with numerous goals, conditional preferences over these goals, a rich library of plans, and preferences over those plans, along with moral values.

Alice is the CEO of a big company. She works at the office every day until 16:45. She has to take the children to school every morning, collect them from school at 17:00, and go have fun with them in the afternoons (until 19:45). Then, they order food at 20:00. Her initial beliefs are her current location, the current weather and time, the current location of her children, whether she owns a car, whether she has worked, if her children have gone to school, if she is at the center of the city, and whether there is a medical emergency.

Table 1 shows her goals, preferences over goals and plans and her values. Alice's library of plans consists of three sets of complex plans: one set of plans for fun-related goals (see left column in Figure 2), one set of plans for transport goals (see right column in Figure 2) and one set for food plans (an extension of the one shown in Figure 1 with an extra plan branch for Chinese food). Goals g_1, g_2, g_4, and g_6 include commuting, and therefore are mapped to transport plans by the metaplanner. g_5 and g_7 are mapped to fun and order meal plans, respectively. The other plans for other goals are trivial: they have a single plan, with a single action (e.g., in the case of the plan to work, there is only one method, with a single action).

Bob is the second agent we have created for this demonstration. Like Alice, he has his own set of beliefs, a place where he lives, a place where he goes to work, preferences over how to have fun, etc. Bob lives in the city center and is a worker in the local factory, every day until 16:45. He has no children so he goes to work directly every morning. Once he is done, he goes to have fun however he prefers. Then he goes back home and orders food at 21:00. His initial beliefs are similar to Alice's, excluding those children-related.

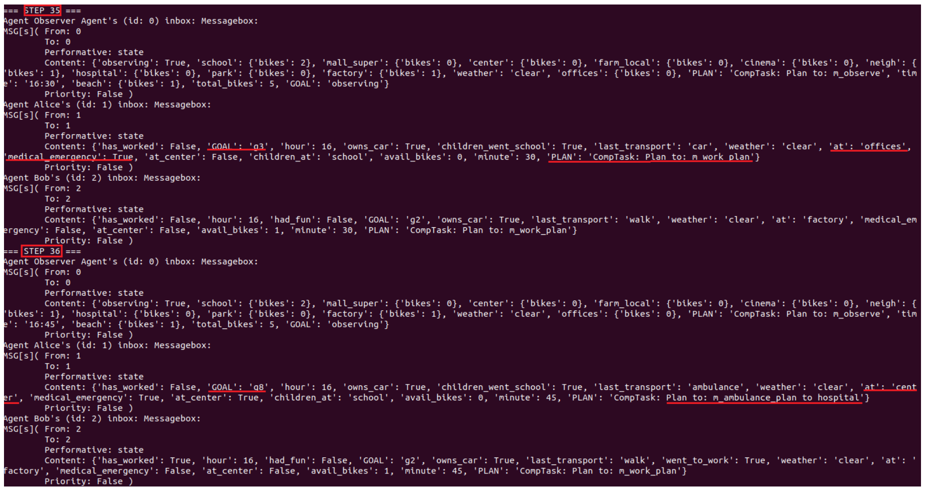
Table 1 shows Bob's goals, preferences over goals, plans and values. Goals g_1, g_3 and g_4 include commuting and therefore are mapped to transport plans by the metaplanner. g_3 and g_5 are mapped to fun and order meal plans, respectively. Bob's goals are a subset of Alice's goals and are mapped to the same plans, but Bob will not act like Alice, as their personal preferences and moral values differ.

4.1 Tests and Results

In this section we show some execution runs to see that agents plan according to their goals, preferences and values, and that they respond to changes in the environment that might cause them to reconsider their contextual preferences and, therefore, need to replan, or even reconsider their goals.

Figure 3 shows the result of a simulation with all default parameters except for `emergencyodds = 0.2` (20%). In it, we can see that Alice is working in her workplace, the offices, when he receives a medical emergency. Then, *her conditional preferences over goals activate*, she changes her current goal, and she rushes to the hospital, as we can see in the next step. Although not shown in the picture, when she goes to the hospital and is cured, her preferences over goals revert to default, and she goes back to the offices to continue working.

In Figure 4 there is the result of a simulation with all default parameters except for `changeodds = 1`, `rainodds = clearodds = 0.5`, and `cloudoods = snowodds = 0`. At step 43, both agents were having fun at the beach. However, it suddenly started to rain, and then *their preferences over plans changed.* The goal (to have fun) does not change. What changes, however, is *how* they decide



```
=== STEP 35 ===
Agent Observer Agent's (id: 0) inbox: Messagebox:
MSG[s]( From: 0
        To: 0
        Performative: state
        Content: {'observing': True, 'school': {'bikes': 2}, 'mall_super': {'bikes': 0}, 'center': {'bikes': 0}, 'farm_local': {'bikes': 0}, 'cinema': {'bikes': 0}, 'neigh': {
'bikes': 1}, 'hospital': {'bikes': 0}, 'park': {'bikes': 0}, 'factory': {'bikes': 1}, 'weather': 'clear', 'offices': {'bikes': 0}, 'PLAN': 'CompTask: Plan to: m_observe', 'tim
e': '16:30', 'beach': {'bikes': 1}, 'total_bikes': 5, 'GOAL': 'observing'}
        Priority: False )
Agent Alice's (id: 1) inbox: Messagebox:
MSG[s]( From: 1
        To: 1
        Performative: state
        Content: {'has_worked': False, 'GOAL': 'g3', 'hour': 16, 'owns_car': True, 'children_went_school': True, 'last_transport': 'car', 'weather': 'clear', 'at': 'offices',
'medical_emergency': True, 'at_center': False, 'children_at': 'school', 'avail_bikes': 0, 'minute': 30, 'PLAN': 'CompTask: Plan to: m_work_plan'}
        Priority: False )
Agent Bob's (id: 2) inbox: Messagebox:
MSG[s]( From: 2
        To: 2
        Performative: state
        Content: {'has_worked': False, 'hour': 16, 'had_fun': False, 'GOAL': 'g2', 'owns_car': True, 'last_transport': 'walk', 'weather': 'clear', 'at': 'factory', 'medical_em
ergency': False, 'at_center': False, 'avail_bikes': 1, 'minute': 30, 'PLAN': 'CompTask: Plan to: m_work_plan'}
        Priority: False )
=== STEP 36 ===
Agent Observer Agent's (id: 0) inbox: Messagebox:
MSG[s]( From: 0
        To: 0
        Performative: state
        Content: {'observing': True, 'school': {'bikes': 2}, 'mall_super': {'bikes': 0}, 'center': {'bikes': 0}, 'farm_local': {'bikes': 0}, 'cinema': {'bikes': 0}, 'neigh': {
'bikes': 1}, 'hospital': {'bikes': 0}, 'park': {'bikes': 0}, 'factory': {'bikes': 1}, 'weather': 'clear', 'offices': {'bikes': 0}, 'PLAN': 'CompTask: Plan to: m_observe', 'tim
e': '16:45', 'beach': {'bikes': 1}, 'total_bikes': 5, 'GOAL': 'observing'}
        Priority: False )
Agent Alice's (id: 1) inbox: Messagebox:
MSG[s]( From: 1
        To: 1
        Performative: state
        Content: {'has_worked': False, 'GOAL': 'g8', 'hour': 16, 'owns_car': True, 'children_went_school': True, 'last_transport': 'ambulance', 'weather': 'clear', 'at': 'cent
er', 'medical_emergency': True, 'at_center': True, 'children_at': 'school', 'avail_bikes': 0, 'minute': 45, 'PLAN': 'CompTask: Plan to: m_ambulance_plan to hospital'}
        Priority: False )
Agent Bob's (id: 2) inbox: Messagebox:
MSG[s]( From: 2
        To: 2
        Performative: state
        Content: {'has_worked': True, 'hour': 16, 'had_fun': False, 'GOAL': 'g2', 'owns_car': True, 'last_transport': 'walk', 'went_to_work': True, 'weather': 'clear', 'at': '
factory', 'medical_emergency': False, 'at_center': False, 'avail_bikes': 1, 'minute': 45, 'PLAN': 'CompTask: Plan to: m_work_plan'}
        Priority: False )
```

Figure 3. Agent Alice changing preferences over goals

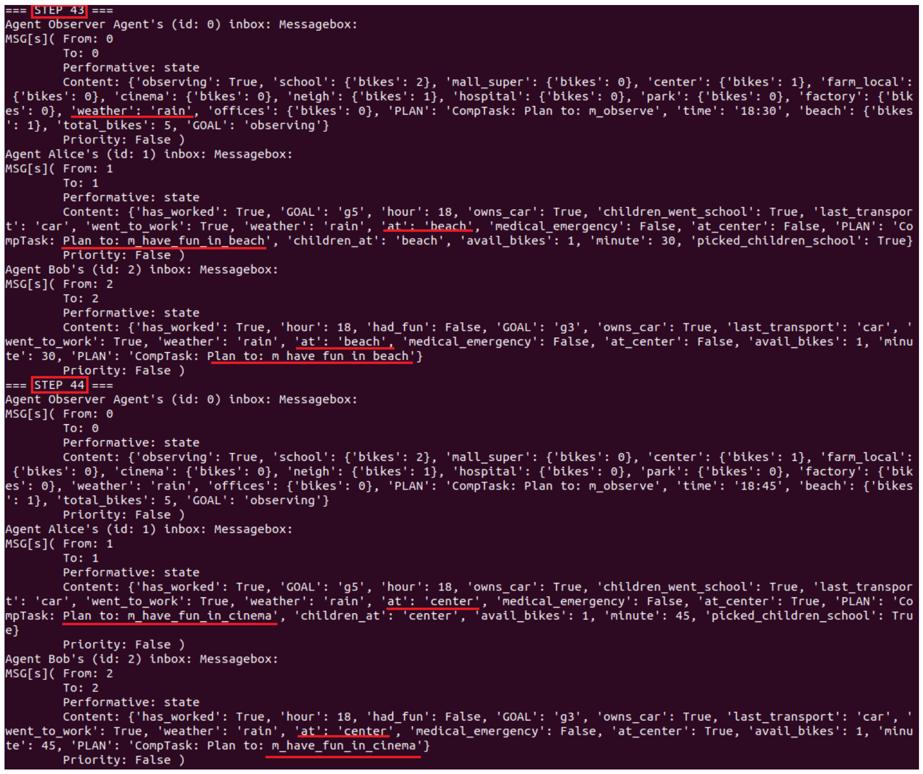
to have fun. Under the sunny weather they preferred to have fun by being at the beach, but when it started to rain they still wanted to have fun and replanned, chosing to go to the cinema instead.

Figure 5 shows an example of the interwork of conditional preferences over plans and values. As we can see, the environment tells us that it is raining. In the case of Alice this triggered a change in her food choice (Chinese) which is fully mandated by her conditional preference over food plans. In the case of Bob, it triggered a change in his transportation means (car) which again is fully mandated by his conditional preference over transportation plans. But his conditional preferences over food are always ordering Pizza above any other option. In the HTN plan related to the "order food" goal (Figure 1), in Bob's case, if he has enough money (more than 12$) he will order pizza (as its permanent, default preference is always pizza first, see Table 1), otherwise he might order falafel. In this case Bob has more than 12$, but there are two possible options to order pizza, and here Bob's values ($local = True$) are used to make the choice. From the two possible options to order pizza, only "order local Pizza" has its local value True and is chosen (see Bob's mental state in Figure 5).

In general, we see that our agents react to changes in their current context by changing their priorities, and always plan according to them. Additionally we see that they function as expected: they pursue their default goals in the correct order, change priorities over goals whenever they should, replan according to changes in both priorities over goals and plans, and make choices based on them.

5 Conclusions

In this paper, we describe an extension to an agent-based simulation environment for High Performance Computing enabling goal-driven agents with hierarchical



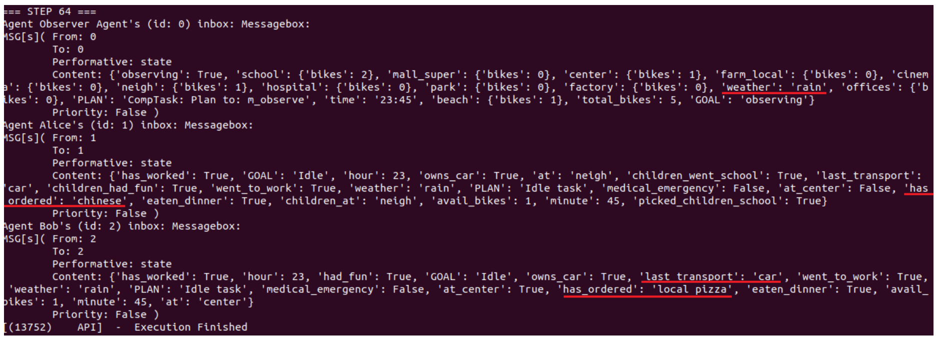
```
=== STEP 43 ===
Agent Observer Agent's (id: 0) inbox: Messagebox:
MSG[s]( From: 0
        To: 0
        Performative: state
        Content: {'observing': True, 'school': {'bikes': 2}, 'mall_super': {'bikes': 0}, 'center': {'bikes': 1}, 'farm_local':
 {'bikes': 0}, 'cinema': {'bikes': 0}, 'neigh': {'bikes': 1}, 'hospital': {'bikes': 0}, 'park': {'bikes': 0}, 'factory': {'bik
es': 0}, 'weather': 'rain', 'offices': {'bikes': 0}, 'PLAN': 'CompTask: Plan to: m_observe', 'time': '18:30', 'beach': {'bikes
': 1}, 'total_bikes': 5, 'GOAL': 'observing'}
        Priority: False )
Agent Alice's (id: 1) inbox: Messagebox:
MSG[s]( From: 1
        To: 1
        Performative: state
        Content: {'has_worked': True, 'GOAL': 'g5', 'hour': 18, 'owns_car': True, 'children_went_school': True, 'last_transpor
t': 'car', 'went_to_work': True, 'weather': 'rain', 'at': 'beach', 'medical_emergency': False, 'at_center': False, 'PLAN': 'Co
mpTask: Plan to: m_have_fun_in_beach', 'children_at': 'beach', 'avail_bikes': 1, 'minute': 30, 'picked_children_school': True}
        Priority: False )
Agent Bob's (id: 2) inbox: Messagebox:
MSG[s]( From: 2
        To: 2
        Performative: state
        Content: {'has_worked': True, 'hour': 18, 'had_fun': False, 'GOAL': 'g3', 'owns_car': True, 'last_transport': 'car', '
went_to_work': True, 'weather': 'rain', 'at': 'beach', 'medical_emergency': False, 'at_center': False, 'avail_bikes': 1, 'minu
te': 30, 'PLAN': 'CompTask: Plan to: m_have_fun_in_beach'}
        Priority: False )
=== STEP 44 ===
Agent Observer Agent's (id: 0) inbox: Messagebox:
MSG[s]( From: 0
        To: 0
        Performative: state
        Content: {'observing': True, 'school': {'bikes': 2}, 'mall_super': {'bikes': 0}, 'center': {'bikes': 1}, 'farm_local':
 {'bikes': 0}, 'cinema': {'bikes': 0}, 'neigh': {'bikes': 1}, 'hospital': {'bikes': 0}, 'park': {'bikes': 0}, 'factory': {'bik
es': 0}, 'weather': 'rain', 'offices': {'bikes': 0}, 'PLAN': 'CompTask: Plan to: m_observe', 'time': '18:45', 'beach': {'bikes
': 1}, 'total_bikes': 5, 'GOAL': 'observing'}
        Priority: False )
Agent Alice's (id: 1) inbox: Messagebox:
MSG[s]( From: 1
        To: 1
        Performative: state
        Content: {'has_worked': True, 'GOAL': 'g5', 'hour': 18, 'owns_car': True, 'children_went_school': True, 'last_transpor
t': 'car', 'went_to_work': True, 'weather': 'rain', 'at': 'center', 'medical_emergency': False, 'at_center': True, 'PLAN': 'Co
mpTask: Plan to: m_have_fun_in_cinema', 'children_at': 'center', 'avail_bikes': 1, 'minute': 45, 'picked_children_school': Tru
e}
        Priority: False )
Agent Bob's (id: 2) inbox: Messagebox:
MSG[s]( From: 2
        To: 2
        Performative: state
        Content: {'has_worked': True, 'hour': 18, 'had_fun': False, 'GOAL': 'g3', 'owns_car': True, 'last_transport': 'car', '
went_to_work': True, 'weather': 'rain', 'at': 'center', 'medical_emergency': False, 'at_center': True, 'avail_bikes': 1, 'minu
te': 45, 'PLAN': 'CompTask: Plan to: m_have_fun_in_cinema'}
        Priority: False )
```

Figure 4. Agents Alice and Bob changing preferences over plans

task network (HTN) plans to choose among goals and among plans based on preferences and a simple moral values model. We have summarized extensions done on the agent model and how they works in a sample scenario. We have also been able to see how 'far' we could go without using any numbers to express preferences over goals, plans, and moral values. As we have seen, we have been able to express conditional preferences over both, have these preferences change based on context, and agents replan based on environmental changes.

One of the biggest limitations from how we declare of goals is that, at any given moment, our agent can only pursue one goal at a time. We cannot have two or more goals active at the same time, and we also cannot 'merge' goals together. This limitation is also common in many BDI-inspired implementations. Only few agent platforms (such as Jason[18] or 2APL[6]) allow to pursue several goals at the same time. We plan to extend the model and the implementation to allow several goals at the same time, specially to allow handling combinations of achievement goals and maintenance goals.

Another limitation is that our agents do not support adding (or removing) goals in runtime. That is, an agent is created and dies with the same set of goals. Goals can be either achieved or not achieved at any given moment, but



```
=== STEP 64 ===
Agent Observer Agent's (id: 0) inbox: Messagebox:
MSG[s]( From: 0
        To: 0
        Performative: state
        Content: {'observing': True, 'school': {'bikes': 2}, 'mall_super': {'bikes': 0}, 'center': {'bikes': 1}, 'farm_local': {'bikes': 0}, 'cinem
a': {'bikes': 0}, 'neigh': {'bikes': 1}, 'hospital': {'bikes': 0}, 'park': {'bikes': 0}, 'factory': {'bikes': 0}, 'weather': 'rain', 'offices': {'b
ikes': 0}, 'PLAN': 'CompTask: Plan to: m_observe', 'time': '23:45', 'beach': {'bikes': 1}, 'total_bikes': 5, 'GOAL': 'observing'}
        Priority: False )
Agent Alice's (id: 1) inbox: Messagebox:
MSG[s]( From: 1
        To: 1
        Performative: state
        Content: {'has_worked': True, 'GOAL': 'Idle', 'hour': 23, 'owns_car': True, 'at': 'neigh', 'children_went_school': True, 'last_transport':
'car', 'children_had_fun': True, 'went_to_work': True, 'weather': 'rain', 'PLAN': 'Idle task', 'medical_emergency': False, 'at_center': False, 'has_
ordered': 'chinese', 'eaten_dinner': True, 'children_at': 'neigh', 'avail_bikes': 1, 'minute': 45, 'picked_children_school': True}
        Priority: False )
Agent Bob's (id: 2) inbox: Messagebox:
MSG[s]( From: 2
        To: 2
        Performative: state
        Content: {'has_worked': True, 'hour': 23, 'had_fun': True, 'GOAL': 'Idle', 'owns_car': True, 'last_transport': 'car', 'went_to_work': True,
 'weather': 'rain', 'PLAN': 'Idle task', 'medical_emergency': False, 'at_center': True, 'has_ordered': 'local pizza', 'eaten_dinner': True, 'avail_
bikes': 1, 'minute': 45, 'at': 'center'}
        Priority: False )
[(13752)    API]  -  Execution Finished
```

Figure 5. Agent Alice having chosen her preferred meal for when it rains, while agent Bob has used his preferred means of transport for when it rains and his "local business" values to choose the local pizza option.

they cannot be eliminated (nor new goals can be added). This limitation was introduced for performance reasons. We plan to tackle this in future extensions.

Perhaps the biggest limitation in our declaration of preferences over goals is that they are absolute, and this stems from the fact that we aimed to not use numbers in our model. Therefore, we cannot express things like 'I prefer this *a little* more than that', or 'I prefer that *a lot* more than this': it is all absolute.

One issue we plan to investigate further is related to what to do when the trigger conditions of non-default preferences over goals overlap. Our current approach is to pick by declaration order, and to allow the designer to implement an ad-hoc, more complex solution, if their scenario requires so. It would be better to modify the structure to allow for a native way to handle this issue.

In the case of our model of preferences over plans, the main limitation is that it is a simplification of Visser's approach, where he provides a more complex structure that allows his agents to have more complex preferences. For instance, his agents can reason about quantities, quantity optimization, limitation by quantity, etc. Also, his agents are able to automatically extract properties of goals by looking at the actions, and then derive the relevant properties of the goals. Our model relies on the designer carefully listing the properties. The limitation regarding overlapping trigger conditions also exists here, as both are handled in the same way.

Finally, our encoding of moral values also totally relies on the designer carefully listing which actions have what moral implications and, while this is good from an expressiveness point of view (it allows us to declare moral relativism as different agents having different moral convictions) and context-dependent morality (the same action carried out under different circumstances having different moral implications), it is a very exhaustive and daunting task. It would be good to have the system partly automated, perhaps employing some matching between the purpose of an action and some pre-existing model of values such as Schwartz's[19], which is employed by Dignum et al. in [15].

References

1. Rousset, A., Hermann, B., Lang, C., Philippe, L.: A survey on parallel and distributed multi-agent systems for high performance computing. Computer Science Review **22** (2016) 27–46
2. Zia, K., Riener, A., Farrahi, K., Ferscha, A.: A new opportunity to urban evacuation analysis: Very large scale simulations of social agent systems in repast hpc (2012)
3. Tisue, S., Wilensky, U.: Netlogo: A simple environment for modeling complexity (2004)
4. Cioffi, C., Sullivan, K.M., Balan, G.C., Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. agent-based modeling of complex crises view project modeling the origins of conflict in east africa view project mason: A multi-agent simulation environment (2014)
5. Lamersdorf, W., Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A short overview middleware view project complex objects view project jadex: A short overview (2011)
6. Dastani, M.: 2apl: A practical agent programming language. Autonomous Agents and Multi-Agent Systems **16** (2008) 214–248
7. Nunes, I., Lucena, C., Luck, M.: Bdi4jade: a bdi layer on top of jade. In: Proceedings of the 9th Workshop on Programming Multiagent Systems. (05 2012)
8. Hindriks, K.V., Roberti, T.: Goal as a planning formalism. Volume 5774 LNAI. (2009) 29–40
9. Abar, S., Theodoropoulos, G.K., Lemarinier, P., O'Hare, G.M.: gent based modelling and simulation tools: A review of the state-of-art software. Computer Science Review **24** (2017) 13–33
10. Gnatyshak, D., Oliva-Felipe, L., Álvarez Napagao, S., Padget, J., Vázquez-Salceda, J., Garcia-Gasulla, D., Cortés, U.: Towards a goal-oriented agent-based simulation framework for high-performance computing. In: Artificial Intelligence Research and Development: Proceedings of the 22nd International Conference of the Catalan Association for Artificial Intelligence, IOS Press (2019) 329–338
11. Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R.M., Torres, J., Cortes, T., Labarta, J.: Pycompss: Parallel computational workflows in python. International Journal of High Performance Computing Applications **31** (1 2017) 66–82
12. Orkin, J.: Three states and a plan: The a.i. of f.e.a.r. M.I.T. Media Lab, Cognitive Machines Group (2006)
13. Bellifemine, F., Poggi, A., Rimassa, G.: Jade - a fipa-compliant agent framework
14. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements (2004)
15. Cranefield, S., Winikoff, M., MVDignum, V.T.D.D., tudelftnl Frank Dignum: No pizza for you: Value-based plan selection in bdi agents (2017)
16. Visser, S., Thangarajah, J., Harland, J., Dignum, F.: Preference-based reasoning in bdi agent systems. Autonomous Agents and Multi-Agent Systems **30** (3 2016) 291–330
17. Kutluhan, E., Hendler, J., Nau, D.S.: Htn planning: Complexity and expressivity. (1994)
18. Bordini, R.H., Hübner, J.F.: Programming Multi-Agent Systems in AgentSpeak using Jason. (2007)
19. Schwartz, S.H.: An overview of the schwartz theory of basic values. Online Readings in Psychology and Culture **2** (12 2012)