Integrating CP-Nets in Reactive BDI Agents

Mohajeri Parizi, M.; Sileno, G.; van Engers, T.

DOI
10.1007/978-3-030-33792-6_19

Publication date
2019

Document Version
Final published version

Published in
PRIMA 2019: Principles and Practice of Multi-Agent Systems

Integrating CP-Nets in Reactive BDI Agents

Mostafa Mohajeri Parizi[1(✉)], Giovanni Sileno[1(✉)], and Tom van Engers[1,2(✉)]

[1] Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
{m.mohajeriparizi,g.sileno,vanengers}@uva.nl
[2] Leibniz Institute, University of Amsterdam/TNO, Amsterdam, The Netherlands

Abstract. Computational agents based upon the belief-desire-intention (BDI) architecture generally use reactive rules to trigger the execution of plans. For various reasons, certain plans might be preferred over others at design time. Most BDI agents platforms use hard-coding these preferences in some form of the static ordering of the reactive rules, but keeping the preferential structure implicit limits script reuse and generalization. This paper proposes an approach to add qualitative preferences over adoption/avoidance of procedural goals into an agent script, building upon the well-known notation of conditional *ceteris paribus* preference networks (CP-nets). For effective execution, the procedural knowledge and the preferential structure of the agent are mapped in an off-line fashion into a new reactive agent script. This solution contrasts with recent proposals integrating preferences as a *rationale* in the decision making cycle, and so overriding the reactive nature of BDI agents.

Keywords: BDI agents · Conditional preferences · Procedural goals · Goal adoption/avoidance · CP-Nets · Reactive agents

1 Introduction

In decision-making and intelligent systems design, when there are multiple ways to achieve a certain goal, the best course of action is usually identified as the one that adheres at best to the user's (or users') preferences. Unexpectedly, current computational models of intentional agents, based upon *belief-desire-intention* (BDI) architectures (e.g. AgentSpeak(L)/Jason [4,21], 2APL/3APL [9,16], GOAL [15]) exhibit a treatment of preferences still relatively underdeveloped with respect to solutions explored in other AI fields like planning or decision systems. All these platforms encode preferences in some form of hard-coded ordering, e.g. of plans, to be used for plan selection. By doing so, the structure of preferences *underlying* such ordering remains implicit, thus limiting transparency and traceability of the choices taken by the modeler, as well as reusability and generalization of the agent scripts (e.g. modifying the preferential structure without modifying the procedural knowledge). Additionally, leaving preferences implicit is particularly problematic if one is targeting institutional design tasks: BDI agents provide a natural model to reproduce behaviours

M. Baldoni et al. (Eds.): PRIMA 2019, LNAI 11873, pp. 305–320, 2019.
https://doi.org/10.1007/978-3-030-33792-6_19

reported in an actual social system, but, without mapping the explicit preferences of their social referents, one cannot make considerations about to what extent a certain policy is affecting individuals.

For these reasons, this paper aims to start reducing the *preference specification gap* for BDI agents, by proposing an extension to the BDI architecture that makes preferences first-class citizens, both w.r.t. representational and computational dimensions. For a similar purpose, Visser et al. [26] have recently proposed a method to integrate preferences as a *rationale* in the decision-making cycle to guide the selection of an intention amongst possible options. However, because the agent looks at its script at execution time, their solution builds upon *reflection*, and so disrupts the reactive nature of BDI agents. In contrast, we propose here a method to *pre-process* (offline) some input procedural knowledge together with an input preferential structure in order to construct a prioritized script. For simplicity, in this paper we will focus only on *procedural goals* and propositional descriptions. In future work, we will consider extensions to first order logic descriptions and declarative goals (achievement and maintenance).

The paper is structured as follows. Section 2 provides an introduction to the BDI architecture and the execution model for reactive agents, an overview of relevant preference representation methods, and presents an extension/modification to the syntax of AgentSpeak(L) to integrate preferences based on CP-nets. Section 3 presents a method to pre-process given procedural knowledge and preferential structure into an agent script. Section 4 presents an example of application. Notes about further developments conclude the paper.

2 Preliminaries

2.1 BDI Architecture and Execution Model

BDI frameworks are usually described in terms of an *agent theory* and an *agent computational architecture* [12]. The agent theory usually refer to Bratman's theory of practical reasoning [7], describing the agent's cognitive state and reasoning process in terms of its *beliefs*, *desires* and *intentions*. Beliefs are the facts that the agent believes to be true in the environment. Desires capture the motivational dimension of the agent, typically in the more concrete form of *goals*, representing procedures/states that the agent wants to perform/achieve. Intentions are selected conducts (or *plans*) that the agent commits to (in order to advance its desires). The agent architecture varies depending on the platform. In Jason [4], for instance, it consists of: perception and actuation modules, a *belief base*, *intention stacks* and an *event queue*. The associated BDI execution model, reproducing the agent's reasoning cycle, can be summarized as follows:

1. observe the external world and update the internal state (perception);
2. update the event queue with perceptions and exogenous events;
3. select events from the event queue to commit to;
4. select plans from the plan library that are relevant to the selected event;
5. select an intended means amongst the applicable plans for instantiation;

6. push the intended means to an existing or a new intention stack;
7. select an intention stack and pull an intention, execute the next step of it;
8. if the step is about a primitive action, perform it, if about a sub-goal post it to the event queue.

As this description exemplifies, an essential feature of BDI architectures is the ability to instantiate plans that can: (a) react to specific situations, and (b) be invoked based on their purpose [22]. Consequently, the BDI execution model naturally relies on a *reactive* model of computation (cf. *event-based programming*), usually in the form of some type of *event-condition-action* (ECA) rules.

2.2 Goal-Plan Rules

A general definition of reactive rules for BDI execution models can be derived from the notion of *goal-plan rules*, i.e. *uninstantiated specifications of the means for achieving a goal* [22], capturing the procedural knowledge (*how-to*) of the agent. A goal-plan rule pr is a tuple $\langle e, c, p \rangle$, where:

- e is the *invocation condition*, i.e. the event that makes the rule *relevant*;
- c, the *context condition*, is a first-order formula over the agent's belief base, which makes the rule *applicable*;
- p, the *plan body*, consisting of a finite and possibly empty sequence of steps $[a_1, a_2, ..., a_n]$ where each a_i is either a *goal* (an invocation attempting to trigger a goal-plan rule), or a *primitive action*.

A goal-plan rule pr_i is an *option* or a *possibility* for achieving a goal-event e, if the invocation condition of pr_i matches with e, and the preconditions pr_i matches the current state of the world, as perceived or encoded in the agent's beliefs. In BDI implementations, the preference between these optional conducts is specified through static rankings assigned by the designer, typically via the ordering of the rules in the code.

Syntax. This paper will refer to a syntax close to that of AgentSpeak(L) [21], introducing a few extensions. If `g` is the name of a *higher-level action*[1], `!g` is a procedural goal (also action-goal or *want-to-do*, usually distinguished from declarative goals/state-goals, or *want-to-be*), that can be referenced to in a plan of action, and `+!g` denotes the goal-event, that acts as triggering event (invocation condition) initiating the commitment towards a plan aiming to perform it. As an example of code, consider:

```
+!g : c <= !a.
+!g <= !b.
```

[1] Higher-level actions are those that can be decomposed in lower-level actions, e.g. the higher-level action "booking a travel arrangement" may have "booking a flight" and "reserving a hotel" as lower-level actions.

The script means that if the triggering event `+!g` occurs, if `c` holds, the agent commits to `a`, otherwise (that is, `c` does not hold) the agent commits to `b`.[2]

For the method proposed here, we will need to refer to conditions concerning adoption and avoidance of certain goals, therefore we extend the previous syntax with new elements for the conditional part of rules: `!g` denotes that `g` is currently adopted and is *active* (i.e. present in the intention stack), and `not !g` states that `g` is *not active* (i.e. absent from the intention stack). Thus, the rule:

```
+!a : !g, not !h, c <= !b.
```

means that when the goal `a` is invoked, if `g` is in the intention stack, `h` is not in it, and condition `c` holds, then the agent adopts the goal `b`.

In the standard syntax, there is no unique identifier to distinguish goal-plan rules (although Jason offers some *labeling* construct). There is also no standard way to have direct access to the plan component of a rule. A possible solution to identify a specific plan without explicit labeling is to refer to the invocation condition of the associated rule alongside its position, e.g. with respect to other rules with the same invocation condition. Consider for instance:

```
+!pay <= !cash.
+!pay <= !credit.
```

Assuming that there is no other rule with the same invocation condition before, the two plans of `cash` and `credit` for the `pay` goal will be respectively denoted as `!pay[0]` and `!pay[1]`. For more clarity, to better separate goal-adoption from the treatment of primitive actions, we will not consider primitive actions as part of preferences (a primitive action `a` will be denoted as `#a`).

2.3 Goal-Plan Graph (Procedural Knowledge)

In the BDI literature, the goal-plan structure expressing the procedural knowledge of an agent is often represented as *and-or* decision trees (see e.g. [8,26]): sequences of sub-goals in each goal-plan rule form the "and" edges (in order to complete that plan, all of the steps should be completed), different goal-plan rules relevant for a goal are the "or" edges (possible plan choices for a given goal). However, presenting the goal-plan dependencies as a tree is a too strong simplification on the possible relations between goals and plans. Procedural knowledge of a BDI agent is often structured by the designer in a manner that plans can be re-used. For example, a `pay` goal can be a sub-goal of any plan concerning buying or reserving something. Further, a tree structure assumes one root goal, when in reality the procedural knowledge structure does not always start from one single goal. Besides, exogenous events may initiate a goal at any level in the goal-plan

[2] Note the backward sense of the arrow "`<=`"; although counter-intuitive with respect to the semantics of production rules, it highlights the underlying backward chaining of instrumental reasoning (the agent commits to `a` because it aims to perform `g`), and consequently, it suggests a priority of evaluation between the rules (the first plan, if applicable, is preferred to the second).

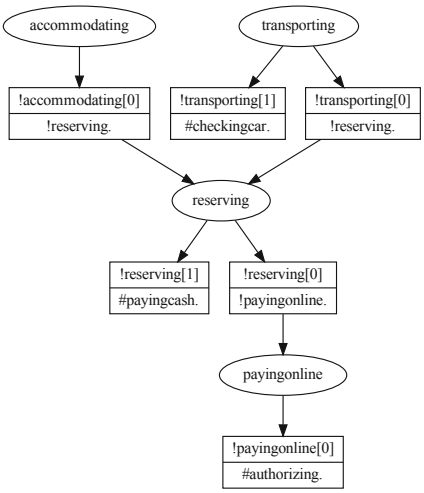

Fig. 1. Example of procedural knowledge illustrated in a goal-plan graph.

graph. This being said, we do assume different levels of granularity of goals and plans, i.e. that there exist higher-level goals/plans and lower-level goals/plans, in the spirit of *hierarchical task networks* (HTNs) [11]. For example, it is not sound having a plan about paying to contain the sub-goal of buying or a plan about preparing for a trip the sub-goal of going for a trip. This assumption is reflected in our representation disallowing loops and recursions in the goal-plan graph. With these constraints, the procedural knowledge of the agent can be modelled as a *directed acyclic graph* (DAG).

Example. Consider the simple script of an agent assisting the user to plan a holiday, e.g. to prepare travel and accommodating. Suppose that two plans are available for the travelling goal: flying and driving; for accommodating, only one: reserving a hotel; the driving plan only contains a primitive action (reminding of checking the car), but the flying plan has the sub-goal of reserving, which is shared with the accommodating plan, etc. The script would be written as:

```
+!travelling => !reserving.
+!travelling => #checking_car.
+!accommodating => !reserving.
+!reserving => !paying_online.
+!reserving => #checking_wallet.
+!paying_online => #authorizing.
```

The associated goal-plan graph is illustrated in Fig. 1 (goals are drawn with ellipses and plans with rectangles). Note how here we slightly modified the AgentSpeak(L) syntax ("`=>`" instead of "`<=`"), to make clear that there is no priority of evaluation between these rules (see note 2). For simplicity, plans of

this running example do not have any context conditions. This is not a limitation: because context conditions only concern the *applicability* of plans, but not their *preferability*, they can be integrated independently to the method that we will present.

2.4 Preference Languages

Several models of preferences have been presented in the computational literature, with various levels of granularity and expressiveness.[3]

The most straightforward *quantitative* approaches are based upon *utility theory* and *decision theory*, under which both planning and action selection problems have shown to be effectively expressed. Typically, by assigning a utility function to each action in each state, the agent/the planner system tries to *maximise* its utility by choosing actions that would result in higher total utility (including avoiding actions with negative utility, e.g. due to cost). The selected plan is called *policy* or *strategy*. Several recent works have investigated the integration of these types of preferences in a BDI architecture. In [10], the authors introduce a utility-based plan selection method triggered at run-time; a similar approach is followed in [8], but here plan selection depends on a given value system, including the case of possibly conflicting values. A hybrid quantitative method is provided by PDDL3 [13], an extension of the *planning domain definition language* (PDDL) [19]. PDDL3 preferences rely on *linear temporal logic* (LTL) over states of the environment. Although based on qualitative descriptions, these preferences are considered quantitative [2] because the valuation of each preference is expressed with a numerical value that corresponds to the number of violations of that preference. This valuation contributes to *preference aggregation strategies* in measuring the quality of a plan, alongside other plan attributes, e.g. resource usage.

While quantitative approaches bring clear advantages in non-deterministic environments or environments with a large state space, they also suffer from the non-trivial issue of translating user's preferences into utility functions. They do not directly support partial ordering and conditional preferences, which are the most natural constructs for humans to express preferences. This explains the existence of a family of *qualitative* proposals. An example of qualitative preference language is LPP [3], relying on first-order, linear temporal logic expressions and *situation calculus* to compute the event dynamics; aggregation is done through different strategy functions, including lexicographic orderings, but also numeric methods. In [25,26], LPP is used to specify preferences about properties of goals and resource usage, and this specification is used during the deliberation phase of a BDI agent.

Other preference models, as CP-nets (qualitative) [5] and GAI networks (quantitative) [14], are introduced specifically for taking into account dependencies and conditioning between preferences in terms of *compact representations* [20]. This is to address the problem of storing preferences in domains with a

[3] For a comprehensive overview (specifically in AI planning), see e.g. [6,17].

large number of features, separately from the problem of choosing from a set of alternative options. Whereas CP-nets have weak constraints, GAI networks build upon the assumption of *generalized additive independence*, and in doing so they enable computing the utility contribution of every single attribute/subset of attributes (they can be seen as the preferential counterpart of Bayesian networks). An additional interesting feature of both CP-nets and GAI-networks is the possibility to be illustrated as intuitive graphical models. To our knowledge, no work has yet attempted to embed these representational models in a BDI architecture. Because they rely on weakest assumptions, and they exhibit primarily a qualitative nature, this work will focus on CP-Nets.

CP-Nets. Conditional *ceteris paribus* preferences networks (CP-nets) are a compact representation of preferences in domains with finite *attributes of interest* [5]. An attribute of interest is an attribute in the world that the agent has some sort of preference over its possible values; in our example, *travelling* can be seen an attribute of interest, with possible values *driving* and *taking a flight*. CP-nets build upon the idea that most of the preferences people make explicit are expressed jointly with an implicit *ceteris paribus* ("all things being equal") assumption. For example, when people say "I prefer to fly rather than to drive", they do not mean at all costs and situations, but that they prefer to fly, all other things being equal to their current situation. CP-nets account also for *conditional preferences* between attributes and their values. An attribute A is said to be the parent of attribute B if preferences over B are conditional over values of A. An example of a preferential system could be "I prefer to go to a close location for holidays, and if I am going to a close location, I prefer to drive, but if I am going to a faraway location, I prefer to fly". Here preference over *location* is unconditional but the preference over *travelling* is conditional, so *location* is the parent of the *travelling* attribute; in practice, *location* is more important than *travelling*. In the graphical presentation of CP-nets, attributes A, B are seen as vertices, and, if A is a parent of B, there is an edge between two attributes A and B.

Syntax. Constraining our attention on procedural goals (*want-to-do*), the preferences we target are about *performance* or *omission* of *higher-order actions*. The attributes of interests for the CP-net are then the possible procedural goals of the agent. Each attribute has two possible values: (1) adoption of the goal, here denoted with the goal name "`!g`", (2) avoiding the goal, denoted as "`not !g`". A preference over performing an action `g` over omitting it in condition `c` can be written as a preference of adopting a (procedural) goal `g` over avoiding it in condition `c`:

```
!g > not !g : c.
```

In general, `c` might be an higher priority preferential attribute or a logical `true` in the case of an unconditional preference.

3 Prioritizing Procedural Knowledge with Preferences

Consider the unconditional preference "I prefer not to do any sort of travel (all the rest being equal)" and the conditional preference "If I am planning a travel, I prefer not to reserve anything". In the previous syntax, this becomes:

```
not !travelling > !travelling : true.
not !reserving > !reserving : !travelling.
```

In this case, the procedural dependency and preferential dependency map well together: in the procedural knowledge graph of the agent, travelling can only precede (i.e. is higher-order with respect to) reserving. At the time the agent is going to plan for reserving, it already knows whether it is reserving a travel by simply checking the presence of this goal in the intention stack. This is not true in the general case: the dependency between preferences may be inverse.

For instance, "I prefer not to pay online, but if I'm paying online, I prefer this to be for paying for a travel (i.e. instrumentally to a travelling goal). I also prefer not to be paying for a travel if it is not by paying online." This preferential structure is written as:

```
not !paying_online > !paying_online : true.
!travelling > not !travelling : !paying_online.
not !travelling > !travelling : not !paying_online.
```

As a procedural dependency, travelling precedes paying online, but as a preferential dependency, online payment is higher than travel. So at the time the agent is choosing to plan a travel, it has not started paying beforehand (either online or cash), that is, the goal of online payment is not yet adopted.

3.1 Plan Meta-data

To deal with this issue, we have considered four sets of *meta-data* for each goal-plan rule pr (and so for each plan p): (1) certain sub-goals as $CG(p)$, (2) possible sub-goals as $PG(p)$, (3) possible intents $PI(p)$ and (4) certain intents $CI(p)$. A certain sub-goal is a goal that will certainly be adopted in all refinements of a plan. A possible sub-goal is one that will possibly be adopted in some refinements of a plan. Possible intents of a plan are all the goals that at some point in their refinement (depending on context) may request the execution of the plan. Finally, we neglect the set of certain intents for a plan because a goal can be adopted for some exogenous event (e.g. an external request), hence, for any goal-plan rule, the only certain intent is the goal appearing in its invocation condition.

3.2 Calculating the Plan Meta-data

In order to calculate these sets, we draw on simple definitions from graph theory. The procedural knowledge is assumed to be an directed acyclic graph $G = \langle V, E \rangle$, where V is the set of vertices (goals and plans), and E the set of edges which either connect goals to plans (*relevant plans*) or plans to goals (*sub-goal*). The set

$r \subseteq V$ consists of root vertices (all goals that are never the sub-goal of any plan), and the set $l \subseteq V$ consists of the leaf vertices (plans that have no sub-goals). Being $v, v' \in V$ two vertices, we introduce three definitions:

- there is a *path* between v and v' is if there is a finite sequence of distinct edges that connect v to v'.
- v is said to *dominate* v' if all the paths from all members of r to v' in the graph pass through v; v is also said to be *dominator* of v'.
- v' is said to *post-dominate* v if all the paths from v to any member of l in the graph pass through v'; v' is also said to be *post-dominator* of v.

Let us assume two goals g and g' with $g \neq g'$ (e.g. `!g1` and `!g2`) and denote with g_i and g'_j plans respectively for g and g' (e.g. `!g1[i]` and `!g2[j]`). The goal g is a *certain sub-goal* of a plan g'_i if g is the post-dominator of g'_i in the goal-plan graph (i.e. all paths from g'_i to a leaf node will visit g). A goal g is a *possible sub-goal* of g'_i if there is a path from g'_i to g and g is not a certain sub-goal of g'_i. A goal g is a *possible intent* of plan g'_i, if there is a path from g to g'_i in the goal-plan tree. By definition, because all g_i plans for a goal g have g as their single shared parent in the graph, then possible intents of these plans are always the same. Finally the only *certain intent* for each plan g_i is g itself.

For instance, w.r.t. our example, the possible intents of `!reserving[0]` are `!accommodating` and `!travelling`, that is, `!reserving[0]` can be executed while attempting to achieve either of these goals; `!paying_online` is a possible sub-goal for the plan `!travelling[0]`, that is, based on the run-time environmental state and contextual conditions, adoption of `!paying_online` may or may not happen in the execution of `!travelling[0]`, etc.

These four sets are mutually exclusive from each other. The set $CI(g_i)$ is mutually exclusive from other sets because $CI(g_i) = \{g\}$ and as a condition for a g' to be in $CG(g_i)$, $PG(g_i)$ or $PI(g_i)$, we have $g \neq g'$. The two sets $CG(g_i)$ and $PG(g_i)$ are mutually exclusive by definition, and $g' \in PG(g_i)$ if $g' \notin CG(g_i)$. Because the procedural graph is assumed to be a DAG, if there is a path from g to g'_i, there cannot be a path from g'_i to g, so we infer that $PI(g_i)$ is mutually exclusive from $CG(g_i)$ and $PG(g_i)$. We have then:

$$CG(g_i) \cap PG(g_i) \cap PI(g_i) \cap CI(g_i) = \emptyset$$

3.3 Rewriting the Agent Script

This section proposes a method to *embed* a given preferential structure into the procedural knowledge of the agent. Informally, the script is rewritten in a manner that the sequential priority between plans follows the explicit CP-net based specification provided by the modeler/programmer. Under the assumption that the preferential structure implied by the CP-net enables an effective *total ordering* of plans and the execution model of the agent dictates that plans are considered in a sequential manner (as in Jason/AgentSpeak(L)), more preferred goal-plan rules will be placed higher in the code. The resulting script is deterministic and keeps the reactivity of the agent intact as it only contains simple conditions on the intentional stack (in addition to contextual conditions).

In essence, the *preferability* of goal-plans is determined by the outcome associated with them. To find the best possible outcome of a goal-plan, the algorithm takes into account what will certainly follow that goal-plan (certain sub-goals), what may follow that goal-plan (possible sub-goals) and creates an outcome for each different run-time motivational contexts in which this goal-plan may be executed (possible intents).

Motivational Contexts. Prior to run-time, there is no certainty about the motivational context of the agent at the time of adopting a goal. To rectify this, we define the set $C(g)$ as the set of all *valid* combinations of the adoption/avoidance of goals present in $PI(g_i)$. The index i is omitted in $C(g)$ because all plans of a given goal share the same intents. Each member of $C(g)$ is representative of a possible motivational context that g may be adopted in. The "valid" qualification means that this combination can occur and this is deduced by the following simple rules. Given two goals $g', g'' \in PI(g_i)$:

- the adoption of g'' can occur together with that of g' if there is a path between them in the goal-plan graph;
- the adoption of g'' can occur together with the avoidance of g', iff g' is not a post-dominator of g'' and g' is not a dominator of g''.

Plan Outcomes. An *outcome* o is an assignment of values (adoption/avoidance) of the variables of W. If we define $Z \subseteq W$, either adoption or avoidance of all $z \in Z$ is present in o. An outcome is a partial outcome if $Z \subset W$ and a complete outcome if $Z = W$. For example, if $W = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ then a possible complete outcome would be $\{\texttt{!a}, \texttt{not !b}, \texttt{!c}\}$. In order to find the preferential priorities between goal-plan rules, the motivational outcomes of each plan should be calculated. As $C(g)$ is the set of possible motivational contexts that a plan g_i may be executed in, we infer the outcome associated to each g_i in each $c \in C(g)$, from here on denoted as $o(g_i, c)$. To calculate the outcome $o(g_i, c)$, we follow the following steps:

1. add all elements of c;
2. add the adoption of all the values of $CG(g_i)$;
3. add the adoption of all the values of $CI(g_i)$;
4. add the avoidance of all variables that are not in other sets, i.e. all p such that $(p \in W) \wedge (p \notin (PG(g_i) \cup CG(g_i) \cup PI(g_i) \cup CI(g_i)))$.

The step 4 captures that all the goals that are not added in the other steps, are impossible to be adopted in the outcome of the plan and so they are considered avoided. Based on the definition of these steps and the mutual exclusion between these sets, after these steps, for each $w \in W$ (all goals) either adoption or avoidance of the w is present in $o(g_i, c)$, except the members of $PG(g_i)$. So o is a partial outcome if $PG(g_i) \neq \emptyset$. This reflects the fact that we can not be sure about the avoidance or adoption of a possible goal in a plan outcome.

Traditionally, an outcome is called *reachable* if an applicable goal-plan rule p exists such that all refinements of p will result in that outcome [27]. However,

as observed in [18], this approach starts from a very pessimistic view, ignoring the fact that the agent itself (not an adversary) chooses which refinement to make, so instead of thinking what it might bring about in all refinements, we are interested in what is the best thing that can happen under some refinement. The best outcome here indicates the optimal outcome according to the preferences specified in the CP-net.

Optimal Outcome. To find the optimal complete outcome $o(g_i, c)$, we use the *forward sweep procedure* presented in [5]. Given the current partial outcome generated from the previous four steps, the method *sweeps* through the CP-net from top to bottom (i.e., from ancestors to descendants), setting each variable that is not present in partial outcome (i.e the members of $PG(g_i)$) to its most preferred valid value (adoption/avoidance) given the values of its parents. This procedure has been proven to return an optimal outcome given the partial outcome as constraint [5]. We only added the condition of validity, to ensure that an outcome can happen. By doing this, for each plan g_i of each goal g, we have a the set $C(g_i)$ representing the possible motivational contexts of g_i and for each $c \in C(g_i)$ we have exactly one optimal complete outcome, as $o(g_i, c)$.

Plan Priorities. At this point, we need to find a best-to-worst ordering between the outcomes of plans of each goal g for each condition $c \in C(g)$. As these are already complete outcomes with respect to the variables of W, we can easily use the *preferential comparison algorithm* presented in [5]. Under a certain condition $c \in C(g)$, given two outcomes $o(g_i, c)$ and $o(g_j, c)$ of two plans g_i and g_j so that $i \neq j$, we say g_i is preferred to g_j if $o(g_i, c)$ is preferred to $o(g_j, c)$, i.e. $g_i \succeq_c^g g_j$ iff $o(g_i, c) \succeq o(g_j, c)$.

Script Rewriting. After computing the $\succeq_c^g$ relation between all plans of each goal g, the script can be rewritten with respect to preferential structure. For all goals for which there is only one plan, no reordering is needed. For each goal g with more than one plan, if $C(g) = \emptyset$, only one ordering is needed and all plans of g are rewritten in the best-to-worst sequence according to $\succeq^g$, alongside their context condition. Otherwise if $C(g) \neq \emptyset$, for each $c \in C(g)$, first the condition over motivational context is written as a logical expression associated to c, encoded as the conjunction of all members of c. Then, same as before, all the plans of g are written in the best-to-worst sequence based on $\succeq_c^g$, possibly with their context condition.

4 Running Example

We illustrate the proposed method on a travel assistant agent, specified with the following preferences:

```
not !travelling > !travelling : true.
not !reserving > !reserving : !travelling.
!reserving > not !reserving : not !travelling.
```

```
!accommodating > not !accommodating : !reserving.
not !accommodating > !accommodating : not !reserving.
!paying_online > not !paying_online : !travelling.
not !paying_online > !paying_online : not !travelling.
```

Table 1. Example of plan meta-data

Plan	CG	PG	PI	CI
`!travelling[0]`	`!reserving`	`!paying_online`		`!travelling`
`!travelling[1]`				`!travelling`
`!reserving[0]`			`!accommodating, !travelling`	`!reserving`
`!reserving[1]`	`!paying_online`		`!accommodating, !travelling`	`!reserving`

The four sets of plan meta-data of the two goals *travelling* and *reserving* from the goal-plan graph of the agent are presented in Table 1. Other plans are omitted because the other goals have only one relevant plan, and then there is not need to rewrite their plans.

For both `travelling` plans, we have $PI(\cdot) = \emptyset$, and so $C(\cdot) = \emptyset$. We calculate the partial outcome based on the rules specified in Sect. 3.3:

$$o(!travelling[0], true) = \{!travelling, !reserving,\ not\ !accommodating)$$
$$o(!travelling[1], true) = \{!travelling,\ not\ !reserving, not\ !accommodating,\ not\ !paying\_online)$$

The outcome for $!travelling[1]$ is already complete. For $!travelling[0]$, based on the CP-net and the partial outcome, the most optimistic substitution for goal *paying_online* is the adoption of this goal, then the complete outcome will be

$$o(!travelling[0], true) = \{!travelling,\ !reserving,\ not\ !accommodating,\ !paying\_online)$$

Comparing these two outcomes with the *improving search* method shows that:

$$o(!travelling[1], true) \succeq o(!travelling[0], true)$$

which in turn gives that

$$!travelling[1] \succeq_{true} !travelling[0]$$

This means that, if the agent has to travel, the driving plan is always preferred and then should always be evaluated before for applicability. The *travelling* plans will be then rewritten as follows:

```
+!travelling <= #checkingcar.
+!travelling <= !reserving.
```

Next, for plans of the goal *reserving*, we have:

$$C(reserving) = \{\{!travelling,\ not\ !accommodating\},\ \{not\ !travelling,\ not\ !accommodating\},\ \{not\ !travelling,\ !accommodating\}\}$$

Considering the first element of $C(reserving)$, denoted as c_1, we have:

$$o(!reserving[0], c_1) = \{!travelling, !reserving,\ not\ !accommodating, !paying\_online\}$$
$$o(!reserving[1], c_1) = \{!travelling, !reserving,\ not\ !accommodating,\ not\ !paying\_online\}$$

Note that both outcomes are already complete: this is because the set PG is empty for both plans. Comparing these two outcomes we can see that:

$$o(!reserving[0], c_1) \succeq o(!reserving[1], c_1)$$

and then we have:

$$!reserving[0] \succeq_{c_1} !reserving[1])$$

This means that, at the point of reserving, if the agent has the intent of travelling but not the intent of accommodating, the plan with online payment is preferred to paying with cash and should be considered first. Finding the ordering for other motivational contexts c_2 and c_3, we will have:

$$!reserving[1] \succeq_{c_2} !reserving[0] \text{ and } !reserving[1] \succeq_{c_3} !reserving[0]$$

and because they result in an equivalent ordering, can be written as

$$!reserving[1] \succeq_{c_2 \vee c_3} !reserving[0]$$

To conclude, the *reserving* plans will be rewritten as follows:

```
+!reserving :
!travelling & not !accommodating
<= !paying_online.
<= #checkingwallet.

+!reserving :
(not !travelling & !accommodating) |
(not !travelling & not !accommodating)
<= #checkingwallet.
<= !paying_online.
```

To simplify the code, we omitted the repetition of the head of the rule for alternative plans associated with the same event/condition coupling, similarly to LightJason [1]. Additional simplifications of the contextual conditions may be obtained by boolean simplification and by considering the sequential evaluation of rules (see e.g. [24] for converting between *constraint-based* and *priority-based* rule-bases).

5 Conclusion and Further Developments

The paper presents an initial contribution towards the integration of the specification of qualitative conditional preferences, expressed as CP-nets, into a BDI agent script. This work focused merely on propositional logic and procedural goals (higher-order actions) and assumed an effective total-ordering on plans. Therefore, as a necessary step towards actual use, the proposal has to be extended in the near future to declarative goals (*achievement goals* and *maintenance goals*) and to *first-order logic* descriptions, and joint with an investigation on how to deal with conflicting partial ordering situations (e.g. forcing total ordering of plans at need calling for user's intervention). Additional investigation is also required for an analysis of the overall algorithmic complexity.

More in detail, for the extension to declarative goals, besides existing proposals in the MAS literature, we are investigating characterizations of *ought-to-do* and *ought-to-be* norms explored in deontic logic, and studies on the interactions between HTN (intuitively related to procedural goals) and STRIPS-like (intuitively related to achievement goals) representations. The core issue we are exploring at the moment concerns how to take into account side-effects, from first principles (primitive actions) to higher-order behavioural constructs, acknowledging that the aggregation of side-effects is non-trivial.

Further, in order to enable an extension to first-order logic, the preferential attitude towards propositional content (which, under a *ceteris paribus* assumption, captures a fully contextualized situation, albeit implicitly) has to be interpreted w.r.t. the internal objects of the proposition and this interpretation seems to bring different results depending on the specific decision-making context (that is, contextual conditions) in which the preference is evaluated; typically it modifies the selection of objects (i.e. in logic programming, it adds additional controls on the unification process), but in certain cases it might entail preparatory actions or sustain maintenance activities. For instance, "when you want to drink during winter, prefer to drink warm drinks rather than cold drinks"; then, if the agent doesn't have yet a drink, it is rational for him to choose a warm drink; but, if he knows already what he'll drink (respectively he is currently drinking), it is rational to attempt to make it warm (resp. keep it warm).

Said that, although the present contribution is a only a first step towards a generally applicable solution, the principle of aiming to a transformation compatible with the reactive nature of BDI agents is a novel technical contribution and sets an important precedent (a higher-level discussion on the separation of reflective from reactive components can be found in [23, Chap. 7]). Replacing

reasoning functions usually implemented with reflective methods with reactive solutions is crucial for many application where computational efficiency is at stake, as for instance model execution for large-scale simulations. In principle, a similar approach could also be extended to other components of the BDI model for which certain authors resorted to reflective methods, like intent selection and event selection [28].

Acknowledgments. This paper results from work done within the NWO-funded project *Data Logistics for Logistics Data* (DL4LD, www.dl4ld.net), supported by the Dutch Top consortia for Knowledge and Innovation Institute for Advanced Logistics (TKI Dinalog, www.dinalog.nl) of the Ministry of Economy and Environment in The Netherlands and the *Commit-to-Data* initiative (commit2data.nl), and partly within the NWO-funded program VWDATA.

References

1. Aschermann, M., Dennisen, S., Kraus, P., Müller, J.P.: LightJason, a highly scalable and concurrent agent framework: overview and application. In: Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2018), pp. 1794–1796 (2018)
2. Baier, J.A., McIlraith, S.A.: On domain-independent heuristics for planning with qualitative preferences. In: AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning, pp. 7–12 (2007)
3. Bienvenu, M., Fritz, C., McIlraith, S.A.: Planning with qualitative temporal preferences. In: Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 134–144 (2006)
4. Bordini, R.H., Hübner, J.F., Vieira, R.: *Jason* and the golden fleece of agent-oriented programming. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming. MSASSO, vol. 15, pp. 3–37. Springer, Boston, MA (2005). https://doi.org/10.1007/0-387-26350-0_1
5. 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. J. Artif. Intell. Res. **21**, 135–191 (2004)
6. Brafman, R., Domshlak, C.: Preference handling - an introductory tutorial. AI Mag. **30**(1), 58 (2009)
7. Bratman, M.E.: Intention, Plans, and Practical Reason, vol. 10. Harvard University Press, Cambridge (1987)
8. Cranefield, S., Winikoff, M., Dignum, V., Dignum, F.: No pizza for you: value-based plan selection in BDI agents. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), pp. 178–184 (2017)
9. Dastani, M.: 2APL: a practical agent programming language. Auton. Agent. Multi-Agent Syst. **16**(3), 214–248 (2008)
10. Deljoo, A., van Engers, T., Gommans, L., et al.: What is going on: utility-based plan selection in BDI agents. In: Proceedings of Workshops at the 31st AAAI Conference on Artificial Intelligence, pp. 711–718 (2017)
11. Erol, K., Hendler, J., Nau, D.S.: HTN planning: complexity and expressivity. In Proceedings of the 12th AAAI Conference on Artificial Intelligence, pp. 1123–1129 (1994)

12. Fisher, M., Bordini, R.H., Hirsch, B., Torroni, P.: Computational logics and agents: a road map of current technologies and future trends. Comput. Intell. **23**(1), 61–91 (2007)
13. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Technical report (2005)
14. Gonzales, C., Perny, P.: GAI networks for utility elicitation. In: Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004), pp. 224–233 (2004)
15. Hindriks, K.V.: Programming rational agents in GOAL. In: El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H. (eds.) Multi-Agent Programming, pp. 119–157. Springer, Boston, MA (2009). https://doi.org/10.1007/978-0-387-89299-3_4
16. Hoeve, E.T., Dastani, M.: 3APL platform. Master's thesis, University of Utrecht, The Netherlands (2003)
17. Jorge, A., McIlraith, S.A.: Planning with preferences. AI Mag. **29**(4), 25–36 (2008)
18. Marthi, B., Russell, S.J., Wolfe, J.: Angelic semantics for high-level actions. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling, pp. 232–239 (2007)
19. McDermott, D., et al.: PDDL - the planning domain definition language. Technical report (1998)
20. Pigozzi, G., Tsoukiàs, A., Viappiani, P.: Preferences in artificial intelligence. Ann. Math. Artif. Intell. **77**(3–4), 361–401 (2016)
21. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Van de Velde, W., Perram, J.W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0031845
22. Rao, A.S., Georgeff, M.P.: BDI agents: from theory to practice. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS 1995), pp. 312–319 (1995)
23. Sileno, G.: Aligning law and action. Ph.D. thesis, University of Amsterdam (2016)
24. Sileno, G., Boer, A., van Engers, T.: A constructivist approach to rule bases. In: Proceedings of the 7th International Conference on Agents and Artificial Intelligence (ICAART 2015) (2015)
25. Visser, S., Thangarajah, J., Harland, J.: Reasoning about preferences in intelligent agent systems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 426–431 (2011)
26. Visser, S., Thangarajah, J., Harland, J., Dignum, F.: Preference-based reasoning in BDI agent systems. Auton. Agent. Multi-Agent Syst. **30**(2), 291–330 (2016)
27. Yang, Q.: Formalizing planning knowledge for hierarchical planning. Comput. Intell. **6**(1), 12–24 (1990)
28. Yao, Y., Logan, B.: Action-level intention selection for BDI agents. In: Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), pp. 1227–1236 (2016)