Implementing Ethical Governors in BDI

Rafael C. Cardoso[1](✉), Angelo Ferrando[2], Louise A. Dennis[1], and Michael Fisher[1]

[1] Department of Computer Science, The University of Manchester, Manchester, UK
{rafael.cardoso,louise.dennis,michael.fisher}@manchester.ac.uk

[2] Department of Computer Science, Bioengineering, Robotics and Systems Engineering (DIBRIS), University of Genova, Genova, Italy
angelo.ferrando@dibris.unige.it

Abstract. Increasingly, BDI agents are being used not just for basic decision-making, but for more abstract ethical decisions. Several authors have built ad-hoc extensions of BDI systems that provide varying levels of sophistication. In this paper, we introduce a general-purpose approach for implementing ethical governors in BDI systems. With this we aim to provide a broad, flexible and consistent framework for implementing increasingly complex ethical reasoning. Our approach is based on a set of domain-independent abstract agents (evidential reasoner, arbiter and execution agent) that together represent an ethical governor. We discuss the implementation of these abstract agents in the Jason agent programming language and demonstrate how they can be used in practice by instantiating agents in two different case studies, one using utilitarianism and the other deontic logic for reasoning about ethical decisions.

Keywords: Ethical governor · Implementing machine ethics · BDI · Jason

1 Introduction

Computational systems can be divided into those which are *implicitly* ethical (in which the process of requirements capture, design and implementation are assumed to guarantee ethical operation of the system), those which are *explicitly* ethical (in which the machine uses some concept of right and wrong as part of its reasoning), and those that are *unethical* [22]. In this paper we take an *explicit* approach to ethical reasoning, in which a machine reasons about the correct course of action by reference to judgements relating to specific ethical principles such as safety, human autonomy and privacy and uses an ethical theory (probably, but not necessarily, from philosophy) to select an appropriate course of action based on those judgements. This is achieved through the use of an

Work supported by UK Research and Innovation, and EPSRC Hubs: EP/R026092 (FAIR-SPACE), EP/R026173 (ORCA), EP/R026084 (RAIN), and EP/V026801 (Verifiability Node). Fisher's work is also supported by Royal Academy of Engineering.

N. Alechina et al. (Eds.): EMAS 2021, LNAI 13190, pp. 22–41, 2022.
https://doi.org/10.1007/978-3-030-97457-2_2

ethical governor that arbitrates decisions, such as plan selection, concerning competing ethical issues. It should not be assumed that in taking this approach we treat the machine as a moral agent in its own right, in our view the morality (and ultimate responsibility) for the machine's behaviour remains with those who commission, design and implement the behaviour – we mean only that the machine's programming explicitly refers to concepts of right and wrong on occasion as part of its functioning.

Belief-Desire-Intention (BDI) [6,23] is a well known model for the implementation of autonomous agents. In this model, the reasoning cycle of an agent revolves around three mental attitudes: *beliefs*, representing the knowledge that the agent has about the world; *desires*, the goals (i.e., state of the world) that the agent wants to achieve; and *intentions*, courses of action that the agent is committed to achieve. In capturing decision-making at this high level of abstraction, the BDI model has the potential to be useful across a range of machine ethics activities, particularly involving the ideas of implementing governors. In particular, the complex reasoning cycle of BDI agents is well suited for performing ethical reasoning, as well as using multiple BDI agents to represent different ethical entities (potentially with opposing/similar views). We use Jason [5], one of the most popular BDI agent programming languages [4,10,20], to implement our approach.

Our approach to implementing an ethical governor in BDI is separated into two levels, abstraction and instantiation. Our main contribution is in the first level, where we introduce three different types of agents (arbiter, evidential reasoner, and execution agent) that implement a communication protocol and together form an ethical governor system. The second level is an instantiation of these types of agents, wherein agents implement the specific behaviours of the application. To evaluate our approach we provide two examples of instantiation, one using utilitarianism and the other using deontic logic as evidential reasoners. Note that while we offer these two types of ethical reasoning by default, our goal with this work is to offer a general-purpose implementation to be used as a basis for experiments with ethical governors that can be further extended with other types of ethical reasoning depending on the requirements of the application.

In this paper, we chose to represent an ethical governor as multiple agents. Other alternatives include representing it as a single agent, or an organisation of agents. Implementing it as multiple agents was more suitable for us given the different types of reasoning that we create to represent the ethical governor (arbiter, evidential reasoners, and execution agent). Using multiple agents allows us to have a clear separation between (potentially conflicting) evidential reasoners (e.g., autonomy vs safety). Moreover, it makes instantiating the agents (i.e., implementing case study specific behaviours) more straightforward, as we can simply create agents that extend their abstract parent and implement only the features necessary in their abstract representation.

This paper is organised as follows. In the next section we discuss some of the background in machine ethics and the related work in implementing it using agents. Section 3 introduces our general-purpose approach to implement ethical governors in the Jason BDI language, explaining its three main elements: evidential reasoner, arbiter and execution agent. In Sect. 4, we evaluate our approach by applying it to two case studies, a remote inspection scenario with a human and a robot cooperating to achieve some goals and a smart home scenario. The paper concludes in Sect. 5 with a summary of our contributions and a discussion about future work.

2 Machine Ethics

Machine ethics is the study of how to implement ethical reasoning in machines. There are a number of approaches to machine ethics, in particular approaches from symbolic artificial intelligence which generally take a philosophical theory and operationalise it, and approaches from machine learning which attempt to learn ethical behaviour from observation. Following [26], symbolic approaches are generally classed as top-down and contrasted to machine learning approaches which are considered bottom-up. There are a number of approaches that seek to combine these, for instance, those in which philosophical theory provides an over-arching framework within which details can be established via learning (e.g., [1]).

Popular philosophical theories for the implementation of explicit machine ethics include utilitarianism (in which the outcomes of actions are scored and the action with the highest score is chosen), deontic logic (in which ethics is encoded as rules that explicitly refer to actions that are permitted, obliged or prohibited in specific situations) [15], and variants on virtue ethics which refer to extent to which an action is in line with some set of desirable values. Many approaches combine aspects of several philosophical theories—such as approaches which evaluate the outcomes of actions with respect to values or *ethical principles* and then use rules to select the preferred choice [1].

One of the earliest implementations of machine ethics is Arkin's *ethical governor* [2]. In this system an ethical governor considers target selection suggestions from autonomous weapon system and reasons about whether the suggestions are compatible with the *Law of War* and the *Rules of Engagement* for a specific situation. The system then vetoes any suggestions that are unethical in these contexts. Ethical governors form a popular class of explicit machine ethics systems where they can act in tandem with more opaque autonomous systems (e.g., deep neural networks) to provide confidence that selected actions are ethical. Among approaches taking inspiration from Arkin's work are those based on the concept of an *ethical consequence engine* [27], in which a simulation engine is used to predict the outcomes of proposed actions which are then passed to a governor style system for evaluation. This relates to Arkin's work in which the governor consists of an *evidential reasoner* and an *application* (which applies either constraint or rule-based reasoning [25] to decide based on the evidence). Further extensions of the ethical consequence engine work have included both the use of BDI style reasoning to arbitrate between choices *and* the ability of the governor "layer" to make its own suggestions for actions if it deems none of those from the underlying system to be acceptable [7]. The work of [7] is here generalised and expanded, in particular to incorporate multiple streams of evidence between which the governor must decide. Our work is more general in comparison to [7], we do not have an explicit robot controller (our approach is not limited to robotic applications), but such a component can be encoded in our execution agent.

In [17], the authors propose an extended BDI architecture where the agent's reasoning is enhanced with case-based reasoning to implement casuistry and consequentialist theories in BDI agents. This is obtained by making the agents use past experiences to solve present problems. In more detail, if a past experience exists, then the agent follows the same steps to solve the problem; otherwise, the agent decides how to solve the problem following the standard BDI flow. Their work is based on a hybrid BDI architecture that uses case-based reasoning while ours solely comprises pure BDI agents. Another extension of the BDI model with the notion of action consequences is proposed in [16]. This is obtained by modelling a consequentialist approach of ethics which makes an agent choose actions with consequences that are less evil. The authors formalise their approach in both Answer Set Programming (ASP) and BDI frameworks. Differently from the works in [16,17], we do not extend the BDI model, instead we present a general-purpose approach to implementing ethical reasoning in (existing) BDI systems without altering the BDI reasoning cycle or the languages and tools that implement it.

The authors in [12] describe a mechanism for BDI agents to have a value-based reasoning process. Such values are used to influence the agent's decision-making, and can relate to ethical aspects. Their approach is similar to ours in the sense that both do not require modifications to the BDI model or to any underlying tool/language. The difference is that in their case their mechanism uses an external constraint solver while ours is directly implemented in the Jason agent programming language. End even though we also have a value-based reasoning process (utilitarianism evidential reasoner), our main focus is in creating the ethical governor system.

3 Ethical Governor in BDI

In [13], Dennis and Fisher note that while in questions of safety such as those studied in the ethical consequence engine work, simulated physical outcomes are effective in evaluating the risks of possible actions when other ethical principles are considered, such as privacy or human autonomy. Evaluating the ethical status of a proposed action might need to reference different processes such as reasoning using rules about possible consequences, simulations of information flows, or reference to stated preferences. Thus a suite of such reasoners is needed, one for each of the ethical considerations in play, and the arbiter (called application in Arkin's work) layer of the evidential reasoner must decide between potentially competing preferences/recommendations/evaluations from these reasoners.

An initial implementation of such a multiple evidential reasoner system using BDI agents programmed in Jason was presented in [9] in which two evidential reasoners – one in the style of the ethical consequence engine that simulated physical outcomes and made recommendations about safety, and one that used its own past history in order to make recommendations about respecting human autonomy – both submitted recommendations to an arbitration system that used utilitarianism to select the desired action. While our previous work was domain-specific, we now present a general approach for implementing machine ethics through an ethical governor system.

Our implementation[1] is written in the Jason [5] agent programming language. Jason started as an implementation of AgentSpeak(L) [24], a theoretical language for BDI systems, but has since seen many extensions such as its use in the JaCaMo [3] multi-agent programming framework. Jason underlying code is implemented in Java and has been shown to have some of the best performance among agent programming languages [21], as well as achieving respectable performance against actor programming languages, especially those that are also implemented in Java [11].

Agents in our ethical governor system consist of:

- an *execution agent*, which is the agent responsible for managing and executing actions that require further ethical reasoning in the system;
- a set of *evidential reasoners*, one or more agents that based on their main characteristic (e.g., autonomy, safety, privacy, etc.) and given an input from the execution agent (i.e., an existing action or a set of states) will suggest an appropriate ethical action;
- and an *arbiter*, in case of two or more evidential reasoners it is necessary that another agent be responsible for determining which action will be sent for execution.

The ethical governor can be part of a larger multi-agent system, including other agents that are not part of the ethical governor (these other agents are out of scope for this paper and are domain-specific). To simplify the implementation details, in this paper we only consider a single ethical governor. To execute an action that requires ethical reasoning, the execution agent asks the evidential reasoners to suggest actions. Actions that require ethical reasoning are identified and defined by the developer of the system. Each evidential reasoner will choose an action based on its instantiation, using epistemic reasoning and its set of beliefs, plans, and Prolog-like rules. Finally, the arbiter collects all suggestions, picks one of them based on an ethical reasoning strategy, such as utilitarianism, and sends the selected choice to the execution agent. Our approach is split into two levels: Abstraction and Instantiation, as represented in Fig. 1.

At the abstraction level, a communication protocol amongst the agents is established. This protocol includes plans for the agents to be able to communicate their requests and replies (using both unicast and multicast), as well as messages introducing the name of the agents that have been instantiated (using broadcast). The code at this level does not require any information about the

[1] Source code available at: https://github.com/autonomy-and-verification/ethicalgov.

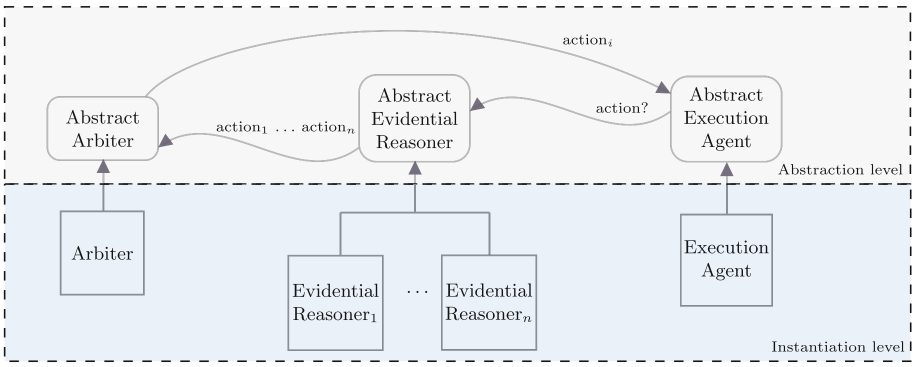


Fig. 1. Overview of our approach for implementing an ethical governor in BDI.

scenario that is being implemented. The abstract level only concerns how agents in the governor system interface with each other.

The instantiated agents *include* their parent abstraction. The internal *include* action in Jason imports at runtime all of the beliefs, goals, and plans of the specified agent source file into another agent source file. This is not the same as the inheritance concept from object-oriented programming, since it is simply loading a preexisting code instead of properly instantiating it. We note here that if we were using the aforementioned JaCaMo multi-agent programming framework [3] we would be able to use Moise [18] (responsible for the organisation layer) to establish an organisation with the roles of arbiter, evidential reasoner, and execution agent, which would allow us to drop the broadcast plans with the name of the agents, since agents in the system would have access to the names of the agents that are playing these roles. However, we opted to have a standalone Jason implementation first since it provides a basic starting point, and leave a JaCaMo extension as future work.

At the instantiation level, we find how the execution agent implements the suggested actions, the implementation logic that the evidential reasoners use to decide which actions to suggest, and finally, how the arbiter weights these suggestions and the type of ethical reasoning that it uses to select the choice that will be sent back to the execution agent.

Even though the abstraction level is at a higher-level and thus more general than the instantiation, both refer to actual implemented code. The code at the abstraction level is simply a parent code that is instantiated and further specialised depending on the application that we want to develop. The instantiation level is further discussed in Sect. 4 when we instantiate the agents using two case studies, but for now we continue to describe the details of the abstraction level.

3.1 The Execution Agent

The execution agent is the agent that will carry out the execution of the action that is selected by the arbiter. In Listing 1, we report the generalised execution agent code. This corresponds to the abstract execution agent from Fig. 1.

The execution agent starts by introducing itself with the addition (represented by the syntax ! preceding a predicate) of a goal at line 1. When the system starts, the addition of this goal generates an event (goal addition event represented by the syntax +!) which triggers the execution of the plan at lines 2–4. The context of the plan (preceded by the : symbol) is used to test if the plan is applicable (i.e., it is the precondition for the plan to be selected). The context at line 3 is always true, since it is used simply to call an internal action that returns the name of the agent and unifies it with the open term `Me`. The plan body (preceded by the `<-` symbol) contains the steps required to achieve a plan, this can be calls to an action, either internal for Jason existing actions or external (provided by the user or the environment), or operations such as goal/belief addition/removal where each call terminates with a semicolon, and finally a dot in the end of the plan. Line 4 calls the internal action `broadcast` to send a message to all agents in the multi-agent system using the `tell` speech act performative, which adds the belief with the name of the execution agent.

```
 1 !introductions.
 2 +!introductions
 3     : .my_name(Me)
 4 <- .broadcast(tell, execution_agent(Me)).

 5 +!act
 6 <-
 7     for (evidential_reasoner(Gov)) {
 8         .send(Gov, achieve, suggest_action);
 9     }.

10 +!choice(ActionList)
11     : .list(ActionList)
12 <-
13     !select_action(ActionList, action(Action, ReasonerType));
14     !execute_action(Action, ReasonerType).
15 +!choice(action(Action, ReasonerType))
16 <-
17     !execute_action(Action, ReasonerType).
```

Listing 1. Generalised execution agent code.

After this step, when the execution agent's instantiation requires an action to be selected by the ethical governor, it first has to call the act plan by adding the `!act` goal (an example of such instantiation, as well as the other agents' instantiation, is shown later in Sect. 4). This plan (lines 5–9) sends a request for an action suggestion to all evidential reasoners through the `achieve` speech act which works as a goal addition, thus triggering the plan `+!suggest_action` in the evidential reasoners when the message is received. The plans shown at lines 10–14 and 15–17 are triggered by a message sent from the arbiter with the action choice that was selected. It is possible for the arbiter to return a list of action choices, in which case the plan at 10–14 is triggered and the instantiation of the execution agent will pick the most appropriate action from the list, and then calls a plan for performing the action where the corresponding callback plan has to be implemented at the instantiation level. Otherwise, if a single action is received the plan at lines 15–17 simply calls the plan for executing it.

It would be possible, with some minor modifications, to remove the execution agent from the abstraction level, however, we would still require an agent to start the process by asking for an action from the evidential reasoners. We have opted to keep this abstraction because it allows the developer to quickly understand which plans have to be implemented in their instantiation (i.e., plans that are domain-specific), without having to resort to any external documentation. Our current implementation of the abstraction level allows for only one instantiation of the execution agent, however, with some minor modifications it should be possible to extend this to allow multiple agents. Namely the communication protocol would have to be extended to include the name of the requesting execution agent, and the evidential reasoners and the arbiter would have to be able to reason about their choices in relation to the requesting agent so that multiple requests could be handled concurrently.

3.2 The Evidential Reasoners

The evidential reasoners are agents that will decide which action to suggest using domain-specific Prolog-like rules that can take into consideration the current state of the system under execution to narrow down which action they believe to be the most suitable for the current situation. When instantiated, these reasoners will often favour diverging opinions, such as for example a safety reasoner in contrast to an autonomy reasoner, as we will see in Sect. 4. In Listing 2, we report the code for the generalised evidential reasoner. This corresponds to the abstract evidential reasoner from Fig. 1.

```
!introductions.
+!introductions
    : .my_name(Me)
<- .broadcast(tell, evidential_reasoner(Me)).

+!suggest_action
    : arbiter(Arbiter) & type(Type)
<-
    !make_choice(Action, Statement);
    .send(Arbiter, tell, evidential_reasoner_choice(Type,
        Action, Statement)).
```

Listing 2. Generalised evidential reasoner code.

Similar to the execution agent, the evidential reasoners also have an introduction plan that works in exactly the same way. The names of the evidential reasoners are necessary for the execution agent, since it needs to ask the evidential reasoners for suggestions, and for the arbiter, since it has to wait each evidential reasoner's choice before selecting one. The plan for handling an execution agent request is defined at lines 5–9, and it is triggered by the message from the execution agent that we have seen in Listing 1. We consult the agent's belief base in the context of the plan (line 6) to unify the name of the arbiter and the type of the agent (which is set in the instantiation level, e.g., safety, autonomy, etc.). The `!make_choice` goal is added at line 8 and it should be triggered by a plan implemented in the instantiation of the evidential reasoner. Its implementation depends on what are the objectives of the instantiated evidential reasoner, which will determine their action choice as well as a statement (e.g., if we are dealing with a utilitarian system than this will be a utility value for that action choice). Finally, at line 9, the evidential reasoner propagates its selection, composed of the type of the evidential reasoner, the action choice, and the statement, to the arbiter of the system. These are then used by the arbiter to select the action to be executed.

3.3 The Arbiter

The arbiter (similar to the entity called application in Arkin's original work) is responsible for collecting the action suggestions from the various evidential reasoners and then selecting the most appropriate based on some ethical reasoning such as utilitarianism. In Listing 3, we report the generalised arbiter code. This corresponds to the abstract arbiter in Fig. 1. Line 1 contains a book-keeping belief `counter(0)` that is used to keep track of how many action suggestions it has received from the evidential reasoners. The introductions plan works the same as in the previous execution agent and evidential reasoners, and it is used by the arbiter to introduce its name to the rest of the agents.

Lines 6–11 and 12–16 contain two plans that receive those choices, both triggered by the addition of the belief `evidential_reasoner_choice`. Both are annotated (preceded by the `@` symbol), a Jason feature that allows plans to have extra information embedded in the plan. In this case, an identifier name and an option that turns the plan into an atomic operation, meaning that the usual concurrent execution of intentions in Jason is stopped once the plan is triggered and will only resume after it has been completed (either with a fail or a success). This is necessary in order to avoid any race condition that could eventually cause the counter belief to be miscalculated. The default plan selection in Jason goes top-down in the plan library of the agent and attempts to select any plan matching the triggering event. Since both plans have the same trigger, the first plan (6–11) will be selected first. Its context checks (using the `.count` internal action which returns the number of times that a particular belief occurs in the belief base) the number of evidential reasoners in the system (beliefs that are obtained through the identification messages) and that the current counter matches this number minus 1 (i.e., this is the last evidential reasoner to send its action choice). The _ symbol indicates variables which may match any value. The body of this plan updates the counter to 0 so that it is ready to receive more action choices in the future and adds the `!arbiter_choice` goal. If the context of the first plan fails, the second one (12–16) will be triggered. Its context is always true, since there will always be a count belief, and its body simply updates the counter by an increment of 1.

```
 1 counter(0).
 2 !introductions.
 3 +!introductions
 4     : .my_name(Me)
 5 <- .broadcast(tell, arbiter(Me)).

 6 @receivelastchoice[atomic]
 7 +evidential_reasoner_choice(Type, Action, Statement)
 8     : .count(evidential_reasoner(_),N) & counter(N-1)
 9 <-
10     -counter(_); +counter(0);
11     !arbiter_choice.
12 @receivechoice[atomic]
13 +evidential_reasoner_choice(Type, Action, Statement)
14     : count(N)
15 <-
16     -counter(N); +counter(N+1).

17 @utilitarian[atomic]
18 +!arbiter_choice : reasoning(utilitarian) & execution_agent
       (Agent)
19 <-
20     +choice(0,0,0);
21     for (evidential_reasoner_choice(Type, Action, Utility)) {
22         -evidential_reasoner_choice(Type, Action, Utility);
23         if (type_multiplier(Type, TypeMultiplier) ) {
24             NewUtility = TypeMultiplier * Utility;
25         } else {
26             NewUtility = Utility;
27         }
28         ?choice(BestUtility, BestType, BestChoice);
29         if (NewUtility > BestUtility) {
30             -choice(BestUtility, BestType, BestChoice);
31             +choice(NewUtility, Type, Action);
32         }
33     }
34     ?choice(Utility, Type, Action);
35     -choice(Utility, Type, Action);
36     .send(Agent, achieve, choice(action(Action, Type))).
```

Listing 3. Generalised arbiter code.

Finally, we have the plan that triggers once all action choices have been received. By default we provide two ethical reasoning mechanisms for the arbiter: utilitarianism and deontic. Other mechanisms can be added in the instantiation of the arbiter as needed. For brevity, we only discuss utilitarianism here (lines 17–36), but show the instantiation of deontic logic later on in our second case study. The context of the plan makes sure that the plan corresponding to the desired ethical reasoning will be selected (utilitarian in this case) and that we know the name of the execution agent. The `+choice` belief is another book-keeping belief to keep track of what is currently the best choice (initialised with 0). We iterate over each choice, and retrieve the scale multiplier for the evidential reasoner (if no multiplier is given in the instantiation of the arbiter, then the utility value is preserved), and use it to update the utility value passed by the evidential reasoner. The scale multiplier can be used to give more (resp. less) importance to certain types of evidential reasoners (e.g., more value to safety rather than autonomy). The best choice is retrieved at line 28, and at line 29 its utility is compared with the utility of the action currently analysed. If the currently analysed action has a greater utility, then the best action is updated (lines 30–31). After all the evidential reasoners' action choices have been evaluated, the best action is retrieved (line 34) and sent to the execution agent (line 36).

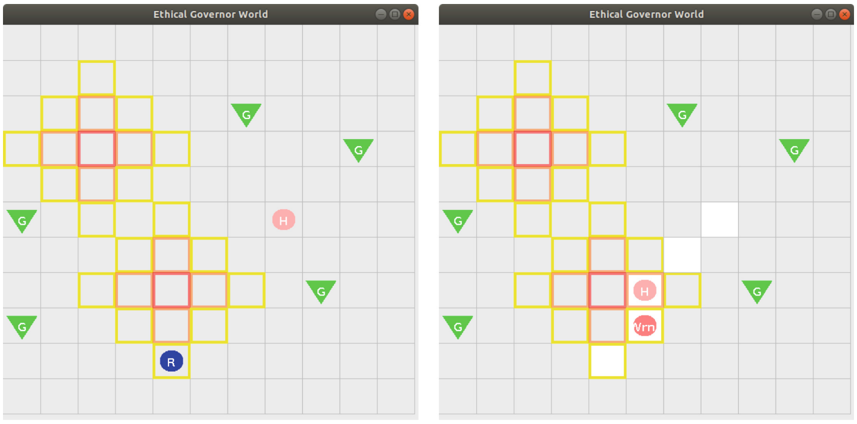
4 Evaluation

To evaluate our general-purpose approach we have selected two case studies and present the instantiation level for both, as well as some results from experimenting with the multiplier scales for different types of evidential reasoners. The choice of the agent to instantiate the execution agent as well as the actions that require ethical reasoning is to be made by the developer of the system.

4.1 Remote Inspection Case Study

Our first case study, shown in Fig. 2, is a simulation where a human (represented by H in the screenshots) and a robot (represented by R in the screenshots) move around in a 2D grid environment. The human's task is to keep visiting all goal positions (green triangles) for as long as the system is running. The robot's objective is to protect the human from stepping into radiation cells; these are the cells with low (yellow), medium (orange), and high (red) level of radiation. The robot has two evidential reasoners which generate the ethical dilemma to be solved by the ethical governor system, a safety reasoner and an autonomy reasoner. In Fig. 2a and b, we report screenshots of the simulated environment. Figure 2a shows the initial configuration. Figure 2b shows the case where the robot is warning the human because the latter is in a dangerous area.

In this case study, we have one execution agent (the robot), two instantiated evidential reasoners (one for safety and one for autonomy), and one instantiated arbiter. The human is also an agent that is part of the simulation, however it does not instantiate any of our abstracted agents. The safety evidential reasoner gives importance to the human safety, preferring actions that will move the robot closer to the human and actions that can warn the human of any imminent danger of radiation. The autonomy evidential reasoner gives importance to the



(a) Initial configuration of the map. (b) Robot warning human.

Fig. 2. Screenshots of the grid map for the remote inspection case study.

human autonomy, preferring actions that will move the robot away from the human, especially when it believes that the human is "annoyed" by its close proximity.

```
 1 { include("evidential_reasoner.asl") }
 2 type(safety).
 3 ...
 4 +!make_choice(Choice, Utility)
 5      : inDanger(human, red) & not near(human, robot)
 6 <- Choice = moveToward; Utility = 3.
 7 ... // the same for orange and yellow but with utility 2
       and 1 respectively
 8 +!make_choice(Choice, Utility)
 9      : inDanger(human, red) & near(human, robot)
10 <- Choice = prevent; Utility = 3.
11 ... // the same for orange and yellow but with utility 2
       and 1 respectively
12 +!make_choice(Choice, Utility)
13      : not inDanger(human, _) & near(human, robot)
14 <- Choice = stayPut; Utility = 1.
```

Listing 4. Instantiation of the safety evidential reasoner.

In Listing 4, we report a snippet of the code for the safety evidential reasoner as an instantiation of the abstract evidential reasoner (the Prolog-like rules such as `inDanger` and `near` as well as some book-keeping beliefs and plans were omitted for brevity). The abstraction of the evidential reasoner is included at line 1 (`.asl` is the file extension for agents in Jason). At line 2, the belief containing the type of the evidential reasoner is explicitly added (this information is required by the arbiter to set the scale multipliers). According to the abstraction in Listing 2, the only plan required to be instantiated is !make_choice. This is the plan that determines the action that will be suggested by the reasoner and sent to the arbiter. At lines 4–14, the three main action options for the !make_choice plan are reported. At lines 4–6 we have the action to move towards a human with utility 3 if the human is in danger (close to a red radiation cell) and the robot is not close to the human to intervene. The action for preventing the human to step in a radiation cell (i.e., issue a warning to the human) is part of the plan at lines 8–10, and it is chosen if the human is in danger (again in relation to a red cell) and the robot is close to the human. There are two extra plans for each of these two actions (move towards and prevent) which are the similar but instead of the red cell the inDanger rule tests for orange and yellow cells and the utility value assigned to these actions is lowered. The last available action for the safety evidential reasoner is a skip action, an action for the robot to stay put (i.e., not move). This is shown at lines 12–14, with the plan being selected if the human is not in any danger and the human and the robot are near each other.

In Listing 5, we report a snippet of the code for the autonomy evidential reasoner. This is similar to the previous code, except that the reasoner is now of the autonomy type and it has two main actions.

```
 1 { include("evidential_reasoner.asl") }
 2 type(autonomy).
 3 ...
 4 +!make_choice(Choice, Utility)
 5     : not near(human, robot) & not annoyed
 6 <- Choice = stayPut; Utility = 1.
 7 +!make_choice(Choice, Utility)
 8     : not near(human, robot) & annoyed
 9 <- Choice = stayPut; Utility = 3.
10 +!make_choice(Choice, Utility)
11     : near(human, robot) & inDanger(human, _) & not annoyed
12 <- Choice = moveAway; Utility = 1.
13 +!make_choice(Choice, Utility)
14     : near(human, robot) & not annoyed
15 <- Choice = moveAway; Utility = 2.
16 +!make_choice(Choice, Utility)
17     : near(human, robot) & annoyed
18 <- Choice = moveAway; Utility = 3.
```

Listing 5. Instantiation of the autonomy evidential reasoner.

At lines 4–6 and 7–9, we have the skip action (stay put) with utilities 1 and 3 respectively; utility 1 is set when the robot is not near the human and the human is not annoyed, while utility 3 is set when the robot is not near the human and the human is annoyed. Annoyed is a rule that checks a proximity belief that indicates for how many consecutive steps has the robot been in a neighbouring cell of the human (any of the 8 positions or even the same position as the human, since there is no collision). If this number is greater than 3 (i.e., the robot has been in close proximity to the human for at least 4 consecutive steps) then the human is considered annoyed. The proximity number is decreased when they are not in close proximity, down to a minimum of 0. At lines 10–12, 13–15, and 16–18, we have the action for the robot to move away from the human with utilities 1, 2, and 3; utility 1 is set when the robot is close to the human and the human is in danger and not annoyed, utility 2 is set when they are near each other and the human is not annoyed, and utility 3 when they are near and the human is annoyed.

The instantiation of the arbiter for this case study is shown in Listing 6. This is the most straightforward instantiation since it relies on the plans from its abstract level representation. As usual, we include the code for the abstract implementation at line 1. At line 2, the kind of reasoning used in the abstract arbiter is set. Since utilitarianism is supported in the abstract implementation we do not need to implement any plans for it. At lines 3 and 4, the utility scale multipliers for the two types of instantiated evidential reasoners are given. In this case, the utilities from the autonomy evidential reasoner are left unchanged, while the utilities of the actions suggested by the safety evidential reasoner are weighted more (20% more). The main idea here is that these values can be customised in order to evaluate the effectiveness of the different evidential reasoners, as we will show later in some of our results for this case study.

```
1 { include("arbiter.asl") }
2 reasoning(utilitarian).
3 type_multiplier(autonomy, 1).
4 type_multiplier(safety, 1.2).
```

Listing 6. Arbiter instantiation.

We do not report the execution agent code (instantiated by the robot) nor the code for the human, since they are not relevant for the presentation of the general technique. The robot contains domain-specific plans which specify how actions such as move away are implemented (move away simply checks the positions of the robot and the human and then selects a cell to move that would bring the robot to be further away from the human) and the human contains plans for moving around the grid efficiently and how to avoid (if possible) radiation cells when warned by the robot. In general, the execution agent could evaluate the arbiter's suggestion, and decide whether to follow it or not. In this case study, the robot executes the action passed by the arbiter without questioning the suggestion. As shown in Listing 7, every different action requires its own plan to be implemented, which could be as straightforward as calling the action directly or could have some other logic such as figuring out which coordinates the robot should move to.

```
+!execute_action(Action, ReasonerType) <- !Action.
```

Listing 7. Execution agent instantiation.

Our approach is made not just as a proof-of-concept, but also to aid in the experimentation of ethical governor systems. In particular, how to fine tune the weights of choices from different types of evidential reasoners. To demonstrate this feature, we have collected several measurements in this case study and observed how they are impacted by changes in the scale multipliers of each type of evidential reasoner. The results are listed in Table 1. For each configuration of the scale multipliers, we ran a simulation cycle of 200 steps. A step is an ordered execution cycle wherein first the robot acts, and then the human can act.

Table 1. Different measurement results for the remote inspection case study when altering the scale multipliers.

Scale Multiplier	Warning	Red Radiation	Orange Radiation	Yellow Radiation	Annoyed	Safety Choices	Autonomy Choices
Safety * 1.2 Autonomy * 1	79	0	1	12	4	166	34
Safety * 3 Autonomy * 1.5	77	0	1	12	10	168	32
Safety * 1 Autonomy * 3	13	0	1	18	3	13	187
Safety * 1 Autonomy * 3.5	0	3	4	11	1	0	200

The warning measurement represents how many times the action `prevent` has been used by the robot. Red, orange, and yellow radiation indicates how many times the human has stepped in one of these cells. The annoyed metric is used to show the maximum number of consecutive steps for which the human was annoyed (i.e., a result of 10 indicates that the human had 10 consecutive steps in which the robot was in close proximity). Finally, safety and autonomy choices are the number of times that choices from these evidential reasoners have been selected by the arbiter. These results show that increasing autonomy has a significant impact in the safety of the human, since it is more likely to step into radiation cells (in particular the dangerous red radiation cells when autonomy has full control). Likewise, increasing safety results in the human being annoyed more frequently, since the robot attempts to follow the human more often, but it does not result in less radiation. This happens because the safety choices for giving out warnings already contain high utility values, thus increasing it has no consequence on the amount of times it issues warnings. These results can then be used to inform the developers in their choice for the most appropriate weights depending on what are the desired outcomes of the system.

4.2 Smart Home Case Study

Our second case study is based on a smart home scenario adapted from the work in [19]. The scenario consists of a smart family home controlled by an intelligent agent. The agent has control over several pieces of smart technology around the house, such as cameras, smart electronics, and an air conditioning system. This system regularly checks the quality of the air in all of the rooms of the home. We simulate a situation where the air conditioning system has detected that there are signs of tobacco smoke in one of the teenager's room. We use our ethical governor system to help the agent come to a decision about what to do when this occurs.

Instead of utilitarianism, we use a simple form of deontic logic wherein the evidential reasoners use epistemic reasoning to select an action and then attach a *yes/no/maybe* recommendation. The arbiter then vetoes the recommendations and instead of sending only one choice as in the utilitarianism example, it sends a subset of them to the execution agent. This behaviour more closely resembles the classical ethical governor architectures found in the literature.

We show this extension of the arbiter in Listing 8, which is simply another plan to be added in Listing 3. Note that this extension is not domain-specific, we simply chose to present it here instead of in the arbiter section for the sake of clarity. The code is straightforward, first the arbiter goes through all of the choices attached with a *yes* recommendation and registers them as choices to be sent to the execution agent by adding a belief `+choice` for each (lines 4–7). If no choices have been added this way, then the arbiter iterates over all the choices marked with a *maybe* recommendation and selects the one associated with the evidential reasoner that has the best rank (i.e., higher priority) among them (lines 8–22). Choices with a *no* recommendation are discarded (lines 23–25) and if no choices were selected by the end then a choice is added with null values (lines 26–28). Finally, the arbiter executes the `.findall` internal action that simply collects all `choice` beliefs and add them to an action list that is then sent to the execution agent.

In this second case study we use a much simpler simulation environment that is used solely to demonstrate another ethical reasoning mechanism. We instantiate six agents, four of which are the *privacy*, *safety*, *legal*, and *reliability evidential* reasoners, as well as the house (smart home execution agent), and the arbiter. The instantiation of the arbiter is almost identical to Listing 6, except that it now uses deontic logic and type ranks. To run our simulation we used the following ranks for the *safety*, *legal*, *privacy*, and *reliability* evidential reasoners respectively: 1, 2, 3, and 4 (lower values mean higher priority). These values can be further optimised as preferred by the developer.

Our simulation starts with the house asking the evidential reasoners what to do after it has detected that there is a teenager smoking tobacco. Each evidential reasoner has one action that it can suggest, along with its recommendation (*yes/no/maybe*). The *legal* evidential reasoner can suggest to warn the authorities with the recommendation: *yes* if tobacco consumption by minors is illegal in the country it is located in and it is not the first time such event occurs and the parents/guardian are not at home, *no* if tobacco is not illegal, and *maybe* if the previous two recommendations fail to be selected. The *privacy* reasoner can suggest to warn the teenager with: *yes* if this is the first time it has detected such behaviour, *no* if this is a repeated occurrence and the parents are at home, and *maybe* if the other two fail. The *safety* reasoner can suggest to warn the parents with: *yes* if the parents are at home, *no* if tobacco is not illegal for minors and the parents are not at home, and *maybe* if the other two fail. The *reliability* reasoner can suggest to log the activity with: *yes* if the log feature is not disabled and the quantity of smoke detected is greater than a certain threshold, *no* if the log feature is disabled, and *maybe* if the other two fail.

```
@deontic[atomic]
+!arbiter_choice : reasoning(deontic) & execution_agent(
    Agent)
<-
    for (evidential_reasoner_choice(Type, Action, yes)) {
        -evidential_reasoner_choice(Type, Action, yes);
        +choice(action(Action, Type));
    }
    if (not choice(_)) {
        for (evidential_reasoner_choice(Type, Action, maybe)) {
            -evidential_reasoner_choice(Type, Action, maybe);
            if (not choice(_) & type_rank(Type, Rank)) {
                +rank(Rank);
                +choice(action(Action, Type));
            }
            elif (rank(BestRank) & type_rank(Type, Rank) & Rank <
                 BestRank  & choice(action(OldAction, OldType)))
            {
                -rank(BestRank); +rank(Rank);
                -choice(action(OldAction, OldType));
                +choice(action(Action, Type));
            }
        }
    }
    for (evidential_reasoner_choice(Type, Action, no)) {
        -evidential_reasoner_choice(Type, Action, no);
    }
    if (not choice(_)) {
        +choice(action(null, null));
    }
    .findall(action(Action,Type), choice(action(Action, Type)
        ), ActionList);
    .send(Agent, achieve, choice(ActionList)).
```

Listing 8. Deontic ethical reasoning plan for the generalised arbiter.

Finally, the execution agent will do nothing if it has received null, or it will select an action from a list of suggestions (with the *yes* recommendation) and execute the selected action, or if it received a single action it will simply execute it. In our instantiation of the execution agent it selects the first action choice from the list, however something more elaborate could be implemented depending on the requirements of the system. Another option would be to allow the execution of all the actions that have been received with a *yes* recommendation, but this would require some minor modifications at the abstraction level of our approach.

Table 2. Actions and recommendations from an example run in the smart home case study. Circled row is the action that the execution agent has chosen.

Reasoner	**Action**	**Statement**
privacy	warn teenager	*yes*
safety	warn parents	*no*
legal	warn authorities	*no*
reliability	log activity	*maybe*

To demonstrate the execution of our approach in the smart home case study we report the results of running a sample configuration of the case study with no control beliefs (e.g., preconditions that check if tobacco is illegal will fail, conversely belief negations will succeed) in Table 2. In this configuration, since the only *yes* recommendation comes from the privacy evidential reasoner, the arbiter agent will discard the others and send that action to the execution agent.

5 Conclusions

In this paper we have described a general approach for implementing ethical governor systems in BDI. Our approach is implemented in the Jason agent programming language and it is divided into two levels: abstraction and instantiation. The abstraction level is domain independent and specifies the standard behaviour and plans of the execution agent, evidential reasoners, and arbiter. Our evidential reasoners and arbiter come equipped with two ethical reasoning mechanisms, utilitarianism and deontic logic. To evaluate our approach we have shown the instantiation of these abstractions using two case studies.

As a future extension of our approach, we intend to modify the action choice output of the evidential reasoners to include a formula containing some default information that can then be used by the arbiter to further augment and inform its selection, in a similar way to the work done in [8]. This formula would contain elements such as why the evidential reasoner believes its choice to be a good choice (i.e., why it has proposed it), the beliefs that it used to come to its conclusion, and any required additional information. This formula would allow us to use different types of reasoning for individual evidential reasoners, for example, a safety evidential reasoner using utilitarianism and an autonomy evidential reasoner using deontic logic. However, the arbiter would also have to be extended to be able to analyse and select an action among these different streams of suggestions, which is a topic that is just recently being researched [14].

References

1. Anderson, M., Leigh Anderson, S.: GenEth: a general ethical dilemma analyzer. In: Proceedings of AAAI 2014 (2014)
2. Arkin, R., Ulam, P., Duncan, B.: An ethical governor for constraining lethal action in an autonomous system. Technical report. GIT-GVU-09-02, Georgia Tech (2009)

3. Boissier, O., Bordini, R., Hubner, J., Ricci, A.: Multi-agent oriented programming: programming multi-agent systems using JaCaMo. In: Intelligent Robotics and Autonomous Agents series. MIT Press (2020). https://books.google.com.br/books?id=GM_tDwAAQBAJ
4. Bordini, R.H., El Fallah Seghrouchni, A., Hindriks, K., Logan, B., Ricci, A.: Agent programming in the cognitive era. Auton. Agent. Multi-Agent Syst. **34**(2), 1–31 (2020). https://doi.org/10.1007/s10458-020-09453-y
5. Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming Multi-agent Systems in AgentSpeak using Jason. Wiley, Hoboken (2007)
6. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press (1987)
7. Bremner, P., Dennis, L.A., Fisher, M., Winfield, A.F.: On proactive, transparent and verifiable ethical reasoning for robots. In: Proceedings of the IEEE Special Issue on Machine Ethics: The Design and Governance of Ethical AI and Autonomous Systems, vol. 107, pp. 541–561 (2019)
8. Bringsjord, S., Sundar, G.N., Thero, D., Si, M.: Akratic robots and the computational logic thereof. In: 2014 IEEE International Symposium on Ethics in Science, Technology and Engineering, pp. 1–8 (2014). https://doi.org/10.1109/ETHICS.2014.6893436
9. Cardoso, R.C., Ene, D., Evans, T., Dennis, L.A.: Ethical governor systems viewed as a multi-agent problem (2020). https://doi.org/10.5281/zenodo.3938851
10. Cardoso, R.C., Ferrando, A.: A review of agent-based programming for multi-agent systems. Computers **10**(2), 16 (2021). https://doi.org/10.3390/computers10020016
11. Cardoso, R.C., Zatelli, M.R., Hübner, J.F., Bordini, R.H.: Towards benchmarking actor- and agent-based programming languages. In: Workshop on Programming Based on Actors, Agents, and Decentralized Control, Indianapolis, Indiana, USA, pp. 115–126 (2013). http://dl.acm.org/citation.cfm?id=2541339
12. Cranefield, S., Winikoff, M., Dignum, V., Dignum, F.: No Pizza for you: value-based plan selection in BDI agents. In: IJCAI, pp. 178–184 (2017)
13. Dennis, L.A., Fisher, M.: Practical challenges in explicit ethical machine reasoning. In: International Symposium on Artificial Intelligence and Mathematics. Fort Lauderdale, USA (2018). http://isaim2018.cs.virginia.edu/papers/ISAIM2018_Ethics_Dennis_Fischer.pdf, also available as arXiv pre-print 1801.01422
14. Ecoffet, A., Lehman, J.: Reinforcement learning under moral uncertainty. CoRR abs/2006.04734 (2020). arXiv:2006.04734
15. Gabbay, D., Horty, J., Parent, X., van der Meyden, R., van der Torre, L. (eds.): Handbook of Deontic Logic and Normative Systems. College Publications, London (2013)
16. Ganascia, J.-G.: Non-monotonic resolution of conflicts for ethical reasoning. In: Trappl, R. (ed.) A Construction Manual for Robots' Ethical Systems. CT, pp. 101–118. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21548-8_6
17. Honarvar, A.R., Ghasem-Aghaee, N.: Casuist BDI-Agent: a new extended BDI architecture with the capability of ethical reasoning. In: Deng, H., Wang, L., Wang, F.L., Lei, J. (eds.) AICI 2009. LNCS (LNAI), vol. 5855, pp. 86–95. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05253-8_10
18. Hübner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. Int. J. Agent-Oriented Softw. Eng. **1**(3/4), 370–395 (2007)

19. Liao, B., Slavkovik, M., van der Torre, L.: Building Jiminy cricket: an architecture for moral agreements among stakeholders. In: Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2019, New York, NY, USA, pp. 147–153. Association for Computing Machinery (2019). https://doi.org/10.1145/3306618.3314257
20. Logan, B.: An agent programming manifesto. Int. J. Agent-Oriented Softw. Eng. **6**(2), 187–210 (2018)
21. Mohajeri Parizi, M., Sileno, G., van Engers, T., Klous, S.: Run, agent, run! architecture and benchmark of actor-based agents. In: Workshop on Programming based on Actors, Agents, and Decentralized Control (AGERE 2020). ACM (2020)
22. Moor, J.H.: The nature, importance, and difficulty of machine ethics. IEEE Intell. Syst. **21**(4), 18–21 (2006). https://doi.org/10.1109/MIS.2006.80
23. Rao, A.S., Georgeff, M.: BDI agents: from theory to practice. In: Proceedings of 1st International Conference on Multi-Agent Systems (ICMAS), San Francisco, USA, pp. 312–319 (1995)
24. 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
25. Shim, J., Arkin, R.C.: An intervening ethical governor for a robot mediator in patient-caregiver relationships. In: Ferreira, M.I.A., Silva Sequeira, J., Tokhi, M.O., Kadar, E.E., Virk, G.S. (eds.) A World with Robots. ISCASE, vol. 84, pp. 77–91. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46667-5_6
26. Wallach, W., Allen, C.: Moral Machines: Teaching Robots Right from Wrong. Oxford University Press Inc., USA (2008)
27. Winfield, A.F.T., Blum, C., Liu, W.: Towards an ethical robot: internal models, consequences and ethical action selection. In: Mistry, M., Leonardis, A., Witkowski, M., Melhuish, C. (eds.) TAROS 2014. LNCS (LNAI), vol. 8717, pp. 85–96. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10401-0_8