






# Embracing AWKWARD! Real-Time Adjustment of Reactive Plans Using Social Norms

Leila Methnani<sup>1</sup>✉ , Andreas Antoniadou<sup>2</sup> , and Andreas Theodorou<sup>1</sup> 

<sup>1</sup> Department of Computing Science, Umeå University, Umeå, Sweden  
{leila.methnani, andreas.theodorou}@umu.se

<sup>2</sup> Guildford, Surrey, UK

**Abstract.** This paper presents the AWKWARD architecture for the development of hybrid agents in Multi-Agent Systems. AWKWARD agents can have their plans re-configured in real time to align with social role requirements under changing environmental and social circumstances. The proposed hybrid architecture makes use of Behaviour Oriented Design (BOD) to develop agents with reactive planning and of the well-established OperA framework to provide organisational, social, and interaction definitions in order to validate and adjust agents' behaviours. Together, OperA and BOD can achieve real-time adjustment of agent plans for evolving social roles, while providing the additional benefit of transparency into the interactions that drive this behavioural change in individual agents. We present this architecture to motivate the bridging between traditional symbolic- and behaviour-based AI communities, where such combined solutions can help MAS researchers in their pursuit of building stronger, more robust intelligent agent teams. We use DOTA2—a game where success is heavily dependent on social interactions—as a medium to demonstrate a sample implementation of our proposed hybrid architecture.

**Keywords:** Reactive planning · Normative agents · Hybrid systems · Multi-agent systems · Games AI

## 1 Introduction

In a Multi-Agent System (MAS) the ability for individual agents to adjust their behaviour when interacting with each other and their environment is critical to the system's success [18]. Yet, agents in MAS need to dynamically re-orient their priorities away from their individual—often selfish—goals and towards the system's collective goals and vice versa as their environment changes.

One technique used to develop agents in highly dynamic environments is Behaviour-Based Artificial Intelligence (BBAI) [11]. Instead of trying to model the environment, BBAI strictly focuses on the actions that an agent can take

---

A. Antoniadou—Independent scholar.

and limiting search within a predefined plan for responsive and robust goal-oriented behaviour [15, 22]. While this approach does indeed increase the search speed, it reduces the flexibility of the system as it is able to react only to *what its developers have specified*. Moreover, BBAI on its own is insufficient when applied to MAS. It does not account for social interactions between agents or any team work explicitly, and thus fails any consideration of real-world challenges where accounting for social behaviours is required.

In this paper, we combine BBAI with formal approaches to get the ‘best of both worlds’ by developing a hybrid architecture: Agents With KnowlEdge About Real-time Duties (AWKWARD). We integrated the OperA framework [19] with Behaviour-Oriented Design (BOD) [15] for their individual and combined strengths in order to produce socially-aware BBAI agents. With OperA, we model the interactions between agents, which contributes towards both governing social behaviour as well as increasing transparency of emerging system behaviour. Transparency could help developers debug the system more effectively and help naive users understand the model [35]. With BOD, we can build reactive planning agents that are suited to interact within uncertain and dynamic environments. We have implemented a ‘toy example’, presented in this paper, for the popular video game DOTA2. Video games have traditionally been used to test AI solutions due to their highly dynamic virtual worlds, which DOTA2 offers as a test bed. Note, we do not consider the specifics of the DOTA2 implementation as our contribution; our focus and contribution is the AWKWARD architecture.

The paper is structured as follows: in Sect. 2, we discuss Behaviour Oriented Design and OperA as the backbone of our architecture, outlining the relevant characteristics of each. In Sect. 3, we introduce the AWKWARD architecture, followed by a sample implementation of the architecture in Sect. 4 and results presented in Sect. 5. In the penultimate Sect. 6, we look at related work done in normative agents, comparing and contrasting those architectures with our own. Finally in Sect. 7, we summarise our contributions and identify future work.

## 2 Background

### 2.1 Behaviour Oriented Design

BOD is a BBAI approach that uses hierarchical representations of an agent’s priorities [15]. These representations express both the priority of the agent in terms of the goals it needs to achieve, and the contexts in which sets of actions may be applicable [12]. Another important feature is the usage of the parallel-rooted hierarchy, which allows for the quasi-parallel pursuit of behaviours and a hierarchical structure to aid the design of the agent’s behaviour. On each plan cycle, the planner alternates between checking for what is currently the highest-level priority that should be active and then progressing work on that priority. Wortham et al. [36] detail the building blocks of a reactive plan in BOD, which are summarised as follows:

1. **Drive Collection (DC):** The root node of the plan’s hierarchy: contains a set of Drives. The DC is responsible for giving attention to the highest priority Drive as at any given cycle only Drive can be active.
2. **Drive:** Allows for the design and pursuit of a specific behaviour. Each Drive has its own release condition of one or more Senses. Even when it is not the focus of the planner attention, each Drive maintains its execution state allowing the quasi-parallel execution of multiple drives.
3. **Competence:** A self-contained basic reactive plan representing the priorities within the particular plan. Each Competence contains at least one non-concurrent Competence Element (CE). Each of these elements is associated with both a priority relative to the other elements and a context which can perceive and report when that element can execute. The highest-priority action that can be executed will do so when the Competence receives attention.
4. **Action Pattern:** Fixed sequences of actions and perceptions used to reduce the design complexity, by determining the execution order in advance.
5. **Action:** A possible ‘doing’ of the agent, such the use of an actuator to interact with the environment; i.e. the means of altering the world and self.
6. **Sense:** A reading of the world or internal status from a sensor of the agent, such as measuring distance between specified units in the world; i.e. the means of reporting environmental and agent status.

BOD aims to enforce the good-coding practice ‘Don’t Repeat Yourself’ by splitting the behaviour into two core modules: the *planner* and the *behaviour library* [14]. The former reads and ‘runs’ the plan at set intervals. The latter contains the blocks of code used by the two primitive plan elements, Actions and Senses. The rest of the plan elements are textually listed in dedicated files, written in Lisp-like format [15], read by the planner. A plan file contains descriptions of both the plan elements and of the connections between the elements.

## 2.2 OperA

OperA is an agent organisation framework for the design and development of MAS consisting of three intermingling models [19]:

1. **Organisational Model (OM):** Describes objectives and the concerns of the organisation from a social perspective. The development of an OM is approached from a top down perspective, that is, with overarching goals and a means to reach them.
2. **Social Model:** Outlines the agent’s role enactment in the form of social contracts. These social contracts describe what capabilities and responsibilities each role demands.
3. **Interaction Model:** Defines interaction agreements between role-enacting agents in the form of interaction contracts. These contracts serve as verification for the fulfilment of interaction agreements between relevant actors specified by organisational objectives as defined in the OM.

OperA requires that all interactions are expressed as *scene* scripts. A scene is a formal specification defining which *roles* within the organisation partake in the interaction, what *landmarks* from the environment indicate the scene’s start, and what resulting signals describe its termination. More importantly, the specification contains a set of *rules* describing the social norms that the participating agents are expected to follow for the scene’s full duration. OperA norms use the deontic expressions of obligation, prohibition and permission as a means of describing an agent’s social behaviour and further validating whether it satisfies or violates organisational expectations.

The OperA framework provides a formal specification that depends on organisational structures and global objectives of the organisation as a whole [19]. Moreover, OperA offers an interaction model between agents without requiring knowledge of the internal architecture of the individual agent itself; this quality in particular is our primary motivation for selecting OperA as a normative MAS framework. Further motivation is offered in Sect. 6 where we compare our selection with other related methodologies.

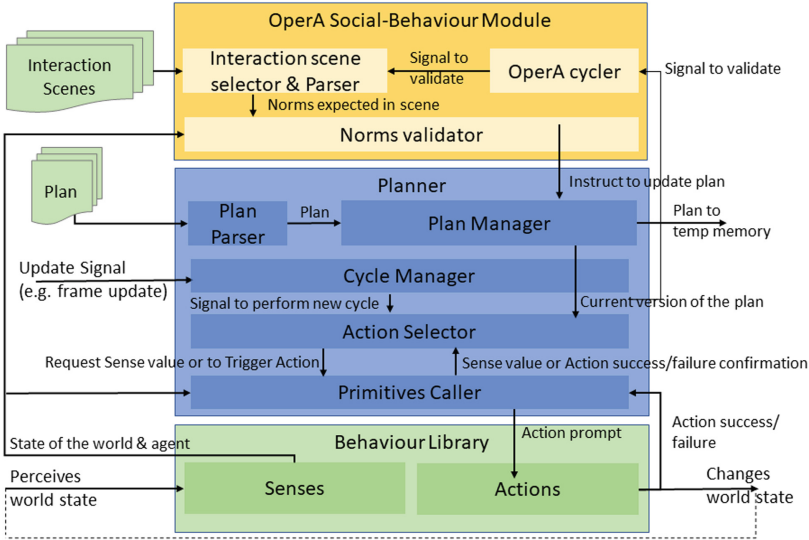
### 3 The AWKWARD Architecture

The AWKWARD architecture, depicted in Fig. 1, is a hybrid-systems architecture designed for agents operating in multi-agent systems. It consists of three modules, each with a distinct purpose: 1) the reactive planner; 2) the OperA module; and 3) the behaviour library. Our solution, inspired by the dual-process theory presented by [26], employs a ‘fast’ system 1 and a ‘slow’ system 2 working in tandem for efficient decision making while taking into consideration its wider environmental and social context.

#### 3.1 The AWKWARD Planner

The ‘fast system’ consists of the reactive planner. The planner allows the agent to act upon its intuitions: plans with multiple drives are triggered based on its environmental and internal changes. Each change may enable short-term or long-term goals for the agent to achieve. Reactive planning has the advantage of faster action-selection and the ability to manage dynamic and unpredictable environments [11, 15]. Most specifically, we use the BOD paradigm due to its proven use in virtual environments, e.g. games [9, 21, 34] and simulations [13]. BOD, unlike other BBAI approaches, allows the execution of multiple behaviours in pseudo-parallel and has a strong emphasis on modularity and reusability.

BOD plans form a hierarchical tree structure that is traversed from the root to the leaves in order of priority. This order determines the agent’s behaviour given the world circumstance it finds itself in. The hierarchy is predetermined by the plan developer and indicated in the plan. In AWKWARD, plans are written as JSON files. At its initiation, the *Plan Parser* component parses the plan to memory, accessed through the *Plan Manager* component, storing the relationships between the plan elements and the hierarchical order of those plan



**Fig. 1.** Conceptual diagram of the AWKWARD architecture. The diagram is colour coded; in yellow, representing our System 1, are the parts of the OperA module, in blue are the components of system 2, i.e. the reactive planner; and in green the code components and files shared during execution by multiple agents. (Color figure online)

elements. Each agent’s DCs can be constructed from the same drive elements, but will differ only in the order of execution, resulting in different expressed behaviours per agent. By initiating all roles with the same plan (i.e. same drive collection hierarchy) and enforcing social norms on agents who violate interaction agreements by explicitly re-prioritising drives, we can shape role- and interaction-dependent plans as needed by the current environment state.

At set intervals, referred to as *ticks*, the planner’s *Cycle Manager* prompts the *Action Selector* component to re-evaluate the agents’ perceived conditions to check if a new plan element needs to be executed or the currently running one should continue doing so. This continuous re-evaluation of the current plan elements, called the *plan cycle*, requires access from the planner to the behaviour library. That is, during each cycle, the planner retrieves the sensory inputs in the form of Sense plan elements, and may trigger actuators in the form of Action plan elements. The plan cycle is set on a fixed frequency based on an external update signal; for example, in our toy implementation, the plan cycle is set on every frame update inline with previous implementations of BOD in games [21].

On every tick, the Action Selector component retrieves the plan from memory and checks the Drive Collections (DCs) in a hierarchical order. If the conditions of a DC are satisfied, as determined by its corresponding Sense elements, it is executed. The planner then traverses through the drive elements of the DC, checking if they are eligible to be executed or not. These comparisons are done by checking by comparing current sensory reading against a set of preconditions,

expressed as sense elements, to determine if the behaviour should be pursued or not by using simple boolean logic. If a drive fires, the planner stops its search at the current tick. This approach of local search enables agents to produce complex behaviours with minimal computational resources as there is no need to explore every possible behaviour at each time [22]. If the drive fired is a different one from the previous cycle, then the existing one ‘pauses.’ In other words, at the DC level, different drives can be in different states of execution enabling a quasi-parallel pursuit of multiple behaviours. Instead, the agent focuses—like our system 1 does—on whatever the highest priority behaviour that should be triggered is, e.g. staying alive, instead of unnecessarily checking if lower priority behaviours could also be triggered [15,36].

### 3.2 The OperA Module

In AWKWARD, the ‘slow’ system is the OperA module. It validates the social behaviour of the agent and provides direction to the reactive planner upon the completion—either with a success or failure—of a drive’s execution. As discussed in Sect. 2, OperA is instantiated with a collection of Interaction Scenes, i.e. formal specification defining which roles within the organisation partake in the defined interaction. Using the senses found in the behaviour library module, the Cycle Manager component prompts the OperA module to check whether a scene has been initiated or terminated.

While a scene is running, the OperA module verifies that the agent’s behaviour fulfils all social obligations that the agent has towards the other agents participating in the same scene. If the agent does not fulfil its obligations, then the OperA module instructs the planner to rearrange the priority of the drives in the currently running Drive Collection. This rearrangement is done by pushing upwards in the hierarchy any drive that corresponds to the desirable social behaviour. The OperA module is informed through formal specifications about which drive should correspond to which social behaviour. It identifies them within the current in-memory version of the plan by using a string equality operation. When the adjustment of drives is done, the new plan is stored in the system’s memory, overwriting the old version. The OperA module checks if the re-prioritisation has produced behaviour that falls within the social and organisational norms the agent needs to comply with. If not, the OperA module continues to adjust the plan’s drives further until the expected social behaviour is achieved. For this social norm validation, OperA uses the same senses as the planner component does to check which plan elements should be executed.

In its past implementations, a single instance of OperA operated at a global level [2,3], i.e. it was responsible for agents and interactions in the model. In AWKWARD, each agent contains its own local instance of OperA; i.e. each AWKWARD agent only checks its own behaviour in its current interaction scene. An advantage of this approach is that it allows us, at least in larger environments, to keep track of the multiple localised interaction scenes that the agent can be in simultaneously.

The use of a localised instance of OperA takes inspiration from Dennett’s description of social constructs, where local efforts are made by agents to “steer their part of the whole contraption while [remaining] blissfully ignorant of the complexities on which the whole system depends” [17]. That is to say, each agent does not need to have any conceivable notion of the global system’s intricacies, but rather a partial comprehension which may suffice for competence. In practical terms, as demonstrated in our toy example that is further discussed in Sect. 4, by having local copies of OperA carried with each agent also means that an AWKWARD agent is not constrained to interacting with other AWKWARD agents only. Rather, the agent can consider and interact with any agent architecture, as well as humans. Moreover, an AWKWARD agent does not require perfect knowledge of all agents in the same environment—which is impossible to maintain in certain scenarios, e.g. competitive games—to continuously adjust and moderate its behaviour within a defined socio-organisational role.

### 3.3 Behaviour Library

While each agent has its own individual plan structure, a single instance of the behaviour library can be shared across all agents. It is the collection of all possible primitives (i.e. Senses and Actions) as discussed in Sect. 2.1. It is accessed through global function calls in the primitives’ tick functions. The behaviour library must therefore maintain a direct pointer to the agent that makes any particular function call in order to return values appropriate to the agent that requested them. By having a single behaviour library for all agents, we not only enable code reusability, but also reduce memory footprint of the agents and support the hosting of a remote behaviour library that runs complex actions and senses [14].

As discussed above, the behaviour library is accessible by both the planner and the OperA modules as they each provide control over the agent’s means of environmental perception, internal status, and any actuators available. This decision to reuse the same Senses enables code reusability but also constrains OperA’s knowledge to that of the agent, which it uses for the action-selection process. Arguably, this is also a more realistic implementation of Kahneman’s dual-system theory [26]: our system 1 acts reactively and system 2 acts deliberately, but the embodiment with its sensors and actuators, i.e. behaviour library, is shared by both systems.

## 4 Implementation in DOTA2

We developed a ‘toy example’ in DOTA2, a popular game characterised by its extremely steep learning curve and complex emerging behaviour through the interactions of the different actors between themselves and their environment. We selected this game due to its complexity, inherent need for a MAS approach, recognition in the AI research community, and access to a public API for AI

researchers<sup>1</sup>. Moreover, DOTA2 has been used in the past by the AI community; OpenAI [7] developed agents that were able to outperform 99% of the DOTA2 player base. Our interest, unlike OpenAI, extends beyond the scope of the individual agent and creating human-beating agents. Instead, we used this highly dynamic environment to demonstrate how AWKWARD can be implemented and have BBAI agents working together in a team by using OperA to model and validate their social interactions.

## 4.1 DOTA2

In the game, two opposing teams of five players (i.e. agents) navigate through terrain, striving to destroy a structure in their enemy’s base known as the Ancient, while also defending their own. There are five roles—or positions—to fill per team. Each team agent is a *hero* that is assigned a position complementary to their given skill-set. For instance, a hero with healing capabilities can fulfil the responsibilities of a *Position 5* role, which includes supporting a hero in the *Position 1* role. In the early game, the Position 1 role is one of the weakest members. As the match progresses, Position 1 typically becomes the strongest on the team. However, to reach that state, a social norm exercised in most games involves giving Position 1 priority, at least in certain scenes, to perform the self-advancing activity known as *farming*. However, given specific circumstances, supporting roles may break that norm to farm in favour of advancing themselves. Hence, it is important to alternate strategies between pursuing behaviours for the benefit of others on the team and for your own personal performance. The resulting team of agents demonstrate emergent behaviour in the arena as they interact. This emergent behaviour can be difficult to perfectly model or even understand from the observer’s perspective.

## 4.2 The Reactive Planner Module and Behaviour Library

As we discussed in the previous section, AWKWARD consists of a planner module and a behaviour library. In order for each agent to behave in compliance with its assigned role, its dedicated plan must consist of a distinctly ordered collection of drives (i.e. DCs). In our DOTA2 implementation, the plan is described in a JSON string that is parsed by the planner. We synchronised the planner update frequency with the game’s internal execution update; i.e. the planner ticks the DC to check its drives on every frame update. In our toy example, we implemented a sample DC with multiple drives.

One of our implemented drives is the *DE-FarmLane*, represented in Formalism 1. The drive prompts the behaviour of seeking out enemy units called *creeps* in the hero’s assigned lane in the environment and striking them when they are low on health in order to achieve what is referred to as a *last hit*. Last hits result in a kill and a gold bounty for the hero to collect. Gold is the currency used to purchase items in-game providing heroes with added attributes and abilities in

<sup>1</sup> Available at: [https://developer.valvesoftware.com/wiki/Dota\\_Bot\\_Scripting](https://developer.valvesoftware.com/wiki/Dota_Bot_Scripting).



battle. For example, a *healing salve* can be purchased and consumed to aid in health regeneration.

The option for health regeneration is captured by Formalism 2. This drive is executed in response to the agent’s internal state: a measure of low health, as defined by the agent designer. It is important to note that two drives such as the external state of *farm time* in DE-FarmLane and internal state of *low health* in DE-Heal, could be true at the same time. In our current implementation, which behaviour is expressed is determined by plan structure; namely the order of executing drives. For example, if DE-Heal is prioritised over DE-FarmLane, then it will execute for as long as the world satisfies its *low health* condition, or until a drive of higher priority is able to run. For instance, if the agent has low health while simultaneously taking damage the plan designer might prioritise a behaviour that more urgently involves retreating.

$$\text{farm time} \Rightarrow \left\langle \begin{array}{l} \text{laning phase ended?} \Rightarrow \text{goal} \\ \text{creep can be last hit?} \Rightarrow \text{lastHitCreep} \\ \text{creep wave far?} \Rightarrow \text{goToCreepWave} \\ \Rightarrow \text{goToAssignedLane} \end{array} \right\rangle \quad (1)$$

$$\text{low health} \Rightarrow \left\langle \begin{array}{l} \text{full health?} \Rightarrow \text{goal} \\ \text{healing ability?} \Rightarrow \text{use healing ability} \\ \text{healing item?} \Rightarrow \text{use healing item} \\ \text{enough gold?} \Rightarrow \text{buy healing item} \\ \Rightarrow \text{retreat} \end{array} \right\rangle \quad (2)$$

The drive element DE-Retreat, represented in Formalism 3, prompts the behaviour of seeking refuge in the occurrence of declining health. This drive, as it is currently implemented, only fires when the hero is below 30% health and directs the hero towards their home base where their *fountain* resides and provides protection while regenerating health.

$$\text{under attack} \Rightarrow \left\langle \begin{array}{l} \text{full health?} \Rightarrow \text{goal} \\ \text{low health AND taking damage?} \Rightarrow \text{retreat} \end{array} \right\rangle \quad (3)$$

Drives may fire Competences or Action Patterns. The competences consist of an ordered list of competence elements that fire other competences or action patterns. The action patterns consist of a sequence of one or more actions—a primitive element found in the behaviour library along with senses. For instance, Formalism 1 depicts DE-FarmLane and lists `lastHitCreep` as a competence element that fires the action pattern shown in Formalism 4. The action patterns consist of `selectTarget` followed by `rightClickAttack`, which are defined in the behaviour library.

$$\langle \text{selectTarget} \rightarrow \text{rightClickAttack} \rangle \quad (4)$$

An agent’s individual behavioural desire is shaped by how its plan element are arranged and executed. Consider again the drive element DE-FarmLane, which

encourages a hero to individually collect as much bounty from last hitting lane creeps as possible. All heroes have this drive in their plans, and while it is a common desire for all members of the organisation to maximise their individual farm, it would not benefit the collective team if all members were to continuously act selfishly. For instance, in the early game a hero with the role of Position 1 is often vulnerable and weak with little of the gold needed to mature their abilities and grow their arsenal. It is therefore the duty of the Position 5 role to sacrifice farm for the sake of their allied member who requires it more at this early stage. Social interactions become particularly interesting and important for us to capture here. This is also where the OperA framework shines, as it enables the definition of social role assignments and interaction agreements for the advancement of the whole team unit.

### 4.3 The OperA Module Implementation

We use social interactions to alter the order in which a reactive agent’s drives are fired. Accomplishing this real-time adjustment should demonstrate successful expression of social behaviour. That is, altering selfish priorities for the collective good of the organisation the individual hero is a member of. The OperA module requires a record of the relevant members and their associated roles within the society. Currently, each hero’s role assignment also determines their lane assignment in the environment, which further characterises their “right”.

For example, Table 1 outlines the role of *Position 1*, indicating that their ultimate objective is to ensure their team’s victory, which can only be done by collecting enough items to become powerful. Their sub-objectives are therefore to farm as much as possible, buying items with earned gold from farm. They have the right to do so in their assigned zone: the *safe lane*—named as such due to its proximity to safety zones. An example norm that applies to a Position 1 hero is the obligation to farm enemy creeps when they are close by. In contrast, Table 2 shows the role table for Position 5, who is not permitted to farm while Position 1 is nearby. This sacrifice is to ensure Position 1 gains enough gold to quickly advance their role and carry the team in the later phases of the game.

**Table 1.** Position 1 role defined using the OperA framework

Role id	Position 1
<b>objectives</b>	Carry team to victory
<b>sub-objectives</b>	Farm and buy items
<b>rights</b>	High priority in safe lane
<b>rules</b>	IF enemy creep around THEN OBLIGED to farm

In the OperA module, relevant norms are constructed by parsing JSON strings with their descriptions. Each norm has a name identifier, an associated

**Table 2.** Position 5 role defined using the OperA framework

Role id	Position 5
<b>objectives</b>	Support team to victory
<b>sub-objectives</b>	Heal allies and take fights
<b>rights</b>	Low priority in safe lane
<b>rules</b>	IF Position 1 nearby THEN NOT PERMITTED to farm

**Table 3.** Interaction Scene for Priority Lane Farming

Scene	Priority lane farming
<b>roles</b>	Carry and support
<b>landmarks</b>	Partner and creeps nearby
<b>results</b>	Partner not nearby
<b>rules</b>	IF highest priority around THEN OBLIGED to farm ELSE NOT PERMITTED to farm

behaviour, and a deontic operator. The behaviour corresponds to the suitable drive element of the agent’s planner. The deontic operators we focus on for this implementation are **NOT PERMITTED** and **OBLIGED**. The **PERMITTED** operator is a softer norm that induces no change to the plan in the current implementation. A norm also has a reference to an assigned agent’s plan. It is validated by checking the agent’s active drive against the expected (norm) behaviour, and whether it is permissible or required within a given circumstance. If a norm is violated, sanctions should be applied. In our implementation, a norm can alter the plan in response to a violation. Recall that our planner implementation allows for the removal and insertion of drive elements at runtime.

While norms can—and should—be associated with the individual agent, what is most interesting for our purposes is the use of norms to characterise interaction scenes between agents. An OperA interaction scene has a unique name identifier, a list of roles involved in the scene, a list of landmarks that indicate the start of a scene, a list of results that indicate the end of a scene, and the rules that indicate the norms that constrain the agents’ behaviours such that they remain within the social expectations of the team (Table 3). In our implementation of scene objects, the landmarks and results correspond to sense primitives that were re-used from the reactive plan elements, as do the rule conditions that determine the appropriate norm to apply in the scene.

Consider again the agent’s **DE-FarmLane** (Formalism 1). This drive is fired when the senses **isFarmingTime** and **isSafeToFarm** return true. We have defined farming time as the early game (approximately the first 10 min.) and safety corresponds to the fight activity and whether any enemy heroes are threatening the agent’s farm ability. For the purposes of this demonstration, we assume it is always safe to farm and the sense will return true. The competence elements

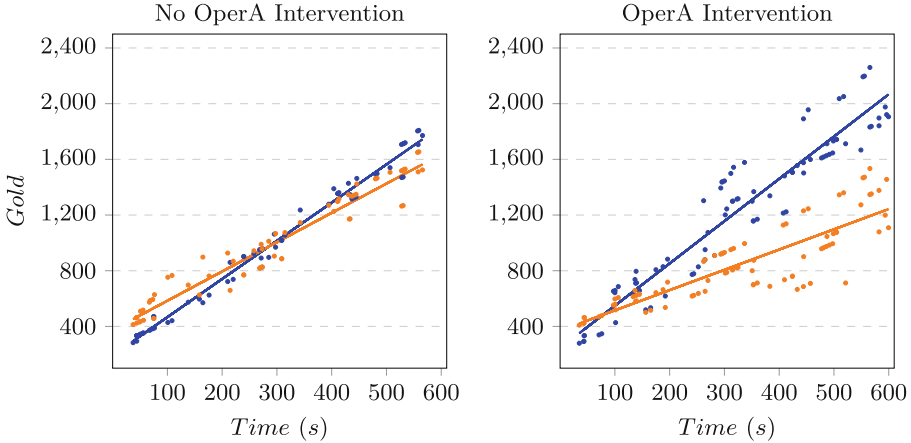
that the farming drive consists of checks conditions in a prioritised order: if the agent is not at the creep wave, they must move there. (Unlike action patterns which always execute in the same order, competences will skip this element if the condition returns false). If the agent is already at the creep wave, then the next condition check is whether any enemy creep around can be last hit. A creep can be last hit when their health is lower than the hero's attack damage and is located within attack range. If this condition is true, the agent will select the appropriate target and attempt to land the last hit. Notice that the drive is entirely self-directed and lacks any social consideration. That is, each agent, while expressing farming behaviour, will pay no mind to whether they have the highest priority around or not. An agent pair farming in the same lane is not optimal behaviour from a social perspective. The agents must abide by social norms for this given circumstance, and OperA can facilitate this interaction using scene scripts. The result is preservation of interaction agreements and altered plans that better suit each agent's role requirements.

While this farm priority check is a simple example that can just as easily be incorporated into each agent's plan, we argue that it will become limiting and wasteful over the course of the game. We observed that when social awareness was incorporated into an individual agent's plan, the structure not only became longer to read and more complicated to understand, but also resulted in increased idle time while an agent was attempting to express farming behaviour. When an agent of higher priority is farming, the agent should not continue to attempt to farm, but should instead fire a different drive for productivity. This coordination is best handled by an organisational/social-aware structure, like OperA. We argue that the individual agent itself should not have too many intricate details about interactions, especially considering the long-time horizons of a game like DOTA2 where social behaviour itself is expected to shift along with the various game phases. In fact, OperA scene scripts very nicely accommodate this game attribute. If behaviour can be altered by OperA, then the reactive planners become simpler to construct. The complexity is captured and described by the OperA interaction models.

## 5 Results

In this paper, we are exclusively concerned with developing a system that can alter agent plans in response to social interactions, regardless of whether other entities in the society have a similar architecture or not. Hence, we designed the evaluation of our sample implementation to reflect that. For the scope of this project, we focus only on the first 10 minutes of the game; within this time in particular, farming priority is important in terms of behaviour adjustment due to expected social norms.

The interaction under evaluation is described by the Priority Farm scene as defined by an interaction scene object. The particular behaviour that is expected is for the Position 5 agent—the AWKWARD bot—to give up its own farm for the benefit of the ally of higher priority in the same lane. In this case, the relevant ally is the Position 1 agent—the default bot.



**Fig. 2.** (Left) Similar rate of gold acquisition between Position 5 (orange) and Position 1 (blue) DOTA2 agents when OperA makes no alterations to Position 5’s plan during the Priority Farm interaction scenes. (Right) Diverging rate of gold acquisition between Position 5 and Position 1 DOTA2 agents as a result of AWKWARD rearranging the plan for the agent in the Position 5 role. Trend lines represent average over  $N = 5$  trials. (Color figure online)

To demonstrate how these changes in the plan impacts the performance of agents, we used gold acquisition as a quantitative metric. This is the standard metric used for players’ performance evaluation in DOTA2 tournaments. The agent’s value of gold is measured over the course of the first 10 min. of the game. The right subplot in Fig. 2 shows the divergence in gold acquisition over time between the AWKWARD bot of position 5 and the default DOTA2 bot of position 1 over five trials. This divergence can be explained by the AWKWARD bot’s social behaviour change; OperA banning farm will result in the role sacrificing its own gain and promoting its ally’s acquisition of gold instead. The AWKWARD bot’s social adjustment is to deny itself from farming when the priority ally (Position 1) is around.

In this scenario, we mark the moments in time where plan changes are expected to occur due to social interactions, but OperA is not inducing the change in order to see the difference in each bot’s gain. While both roles still acquire gold, Position 1 has acquired less than expected, and both agents approximately match each other’s gain.

In contrast, the left subplot in Fig. 2 illustrates a different trend when the AWKWARD bot does *not* change its plan within the social context. These plots show the two roles on par with one another in terms of gold acquisition over time. When the AWKWARD bot does not alter its plan and continues to attempt to farm, even while the ally of higher priority (Position 1) is also farming in the same lane. The difference in gold acquisition between scenarios can be seen over numerous trials (the dotted lines shows the average linear trend over 5 trials).

The data varies in time and gold value due to added randomness across game instances, but the overall trends remain similar.

## 6 Related Work

The AWKWARD architecture combines the normative framework OperA with the Behaviour-Based AI architecture BOD. In this section, we review and compare our architecture to relevant approaches found in the literature. As a complete survey of the literature is beyond the scope of this article, we focus on the most relevant approaches that we considered during the conceptualisation of the AWKWARD architecture.

### 6.1 Behaviour-Based AI

Various approaches of Behaviour-Based AI have been proposed for achieving real-time performance in embodied—physical or virtual—agents including the Subsumption Architecture [10], Pengi [1], and ANA [29]. These bottom-up reactive planning approaches use condition-action pairs without—or with minimal—internal state; i.e. a simple functional mapping between perceived environmental stimuli and their appropriate responses. Such reactive approaches have proven highly effective for a variety of problems [30]. However, in comparison to BOD, these approaches expect little regularity in the arbitration of behaviour; i.e. all possible behaviours must be considered at all times. Moreover, they cannot store information dynamically, thus putting the onus on the developer to predict possible stimuli and develop the appropriate behaviours. BOD overcomes these issues by further decomposing behaviours into “things that need to be checked regularly, things that only need to be checked in a particular context, and things that one can get by not checking at all” [15].

While BOD also originated as a robotics cognitive architecture with its POSH implementation [15]—and most recently Instinct [36]—it has made its way to virtual environments such as games, e.g. pyPOSH in Unreal Tournament [9], POSH-Sharp in StarCraft [21], and UNPOSH in Unity [34]. However, in these implementations of BOD, there was no mechanism to verify that the agents adhere to their social roles. Instead, unlike our AWKWARD implementation, the plan developer had to ensure that any social norms were accounted for during the plan’s development, limiting the possible interactions between agents and team-level performance of the agents.

### 6.2 Normative Agents and Self-organisation

The AWKWARD architecture is related to previous work done in the development of *normative agents*. Early work in the literature for normative agents proposes architectures that extend Belief, Desire, Intention (BDI) models with norms [16, 27, 28]. These agents deliberate around the generation and selection of both goals and plans. For instance, [8] describe their (Beliefs, Obligations,

Intentions, and Desires) BOID agent architecture that appends obligations as a mental attitude in addition to its beliefs, desires and intentions. However, as [20] argue, BDI agents focus towards their own goals instead of social interactions—both with other artificial and human agents—making such approaches unsuitable for multi-agent systems where cooperation between agents is necessary.

Another related approach is N-Jason, a norm-aware BDI agent interpreter equipped with the programming language for agent norm compliance at runtime [28]. It extends Jason/AgentSpeak(L) with addition of normative concepts such as obligations, permissions, prohibitions, deadlines, priorities, and duration. Similar to N-Jason, N-2APL is also a BDI agent architecture that supports norm-aware deliberation [4]. N-2APL allows agents to adopt normative behaviour in the form of deontic obligations and prohibitions with specified deadlines. OperA being a framework for formal specifications of social interactions instead of a complete architecture, enables us to define explicitly the social interactions in the form of scene specifications while also keeping the reactive planner component independent; i.e. operational without the OperA module.

Relevant work also includes existing methodologies for the development of normative agents in multi-agent organisations such as Moise [23] and its extension Moise+ which adds an inheritance on the roles and structural verification features [24, 25]. Moise+ is based on notions of *roles*, *groups*, and *missions*, enabling explicit specification of MAS organisations that agents can reason about and organisational platforms can enforce [25]. Moise+ offers an implicit description of an interaction protocol through deontic links that specify agent permissions and obligations within their assigned missions. An agent belongs to groups where they are offered a set of permitted roles and missions. Thus, upon changing roles and missions, groups are also subject to change, allowing for task-oriented coalitions to be defined. While interactions in Moise+ are task-driven, OperA leads by social expectation in the form of explicit contracts; a main motivation for its adoption in the work presented here.

All of these approach capture and represent social norms in order to enable agents to self organise. However, they rely on integrating the social norm enforcement directly into the decision making system. In our architecture, we allow for both reactivity and social deliberation as the situation demands. This ensures that our agents can act efficiently in their environment on their own, i.e. act as complete complex agents, while also organising themselves based on their social roles—and corresponding responsibilities—when they are a part of a larger organisation. Moreover, our decision to use a distributed version of OperA ensures that our AWKWARD agents can interact with other AWKWARD agents, non-AWKWARD agents, and even humans.

Finally, AWKWARD considers that all agents have the capacity to act selfishly and altruistically to varying degrees determined by their roles and social interactions. In contrast to above approaches, we use a reactive planning architecture for the individual agent motivated by its ability to handle uncertainty and dynamic environments. We assign obligations via external expectations that may be subject to change. This approach of global coordination also differs

from other work proposed for developing normative reactive planning agents, such as the NoA architecture [27]. While NoA adopts norms and deliberates over their activation, our proposed framework concerns itself with dynamically imposing, monitoring and enforcing norms through global coordination, and then distributed enforcement, rather than individual deliberation.

### 6.3 Hybrid Approaches with Reactive Planning

AWKWARD combines formal reasoning, in the form of the OperA module, with reactive planning. A related approach is the logic-based subsumption architecture, where the different control layers that make up a subsumption system have been axiomatised using first-order logic [5,6]. The benefit of this approach is the introduction of non-monotonic reasoning into reactive planning; i.e. developers can understand and easily add control layers to the subsumption planner at run time [6]. Similarly, the Layered Argumentation System combines—at hardware level—fuzzy reasoning and non-monotonic reasoning for run-time generation of reactive plans [32,33]. These combinations of reactive and formal reasoning approaches bridge together communities—one of our goals—but their focus has been to improve the overall performance of the agent by combining the two paradigms instead of enabling cooperation between multiple agents.

With the latter goal in mind, ABC<sup>2</sup> architecture combines classical planning with reactive approaches [31]. ABC<sup>2</sup>, similar to AWKWARD, emphasises cooperation between agents. In ABC<sup>2</sup> each agent has to define and broadcast their own ‘skills’; i.e. it requires active communication and coordination instead of promoting self-coordination as AWKWARD’s OperA implementation does. AWKWARD keeps the reactive and formal parts of the system completely separate from each other; i.e. our reactive planner—sans social consideration—can operate without the OperA module.

AWKWARD uses both formal reasoning and reactive planning as complementary approaches to each other. However, unlike past approaches, this combination is done on an architectural level. By keeping the reactive planner separate from the normative system, our reactive planner can operate without the normative reasoning module—even if the agent is not behaving within the limits of its socio-organisational role.

## 7 Conclusions and Future Work

In this paper, we presented the AWKWARD architecture for hybrid systems: agents that combine normative reasoning, in the form of OperA, and behaviour-based AI methods, in the form of BOD, at an architectural level. Combining the advantages of BOD and OperA, AWKWARD achieves real-time adjustment of agent plans for evolving social roles as it verifies—and adjust plans to ensure—adherence to any socio-organisational role prescribed to the agent. We provided a toy example, implemented in the game DOTA2, where we demonstrated how



AWKWARD enables continual manipulation of agent’s behaviour over changing environmental and social circumstances.

With the planner and OperA module implemented in DOTA2, we have demonstrated how OperA can influence the behaviour of reactive planners under defined social circumstances. However, manually designing and building reactive plans can be inefficient and time consuming for system developers, especially as plans scale. As a next step, we intend to investigate methods of optimising plan structure automatically. To do this, Reinforcement Learning techniques can be employed to guide the discovery of an optimal ordering and OpenAI Gym acts as a favourable toolkit for this task. Additionally, we are interested in extending the scope to cover variable autonomy, where varying levels of decision-making control can be passed between a human player—who either directly controls a hero in the game or oversees a team of bots—and artificial agents.

**Acknowledgements.** Theodorou was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We would like to thank Dignum V. for her input on OperA. All code is available at: <https://github.com/lulock/dota>.

## References

1. Agre, P., Chapman, D.: Pengi: an implementation of a theory of activity. In: Proceedings of the Sixth National Conference on Artificial Intelligence (1987)
2. Aldewereld, H., Dignum, V.: OperettA: organization-oriented development environment. In: Dastani, M., El Fallah Seghrouchni, A., Hübner, J., Leite, J. (eds.) LADS 2010. LNCS (LNAI), vol. 6822, pp. 1–18. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22723-3\\_1](https://doi.org/10.1007/978-3-642-22723-3_1)
3. Aldewereld, H., Dignum, V., Jonker, C.M., van Riemsdijk, M.B.: Agreeing on role adoption in open organisations. *KI-Künstliche Intelligenz* **26**(1), 37–45 (2012)
4. Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents. In: Proceedings of the 11th International Conference on Autonomous Agents and Multi-agent Systems, vol. 2, pp. 1057–1064 (2012)
5. Amir, E., Maynard-Reid, P.: Logic-based subsumption architecture. In: *Artificial Intelligence* (2004)
6. Amir, E., Maynard-Zhang, P.: LiSA: a robot driven by logical subsumption (2001)
7. Berner, C., et al.: DOTA2 with large scale deep reinforcement learning. arXiv preprint:1912.06680 (2019)
8. Broersen, J., Dastani, M., Hulstijn, J., van der Torre, L.: Goal generation in the BOID architecture. *Cogn. Sci. Q.* **2**, 428–447 (2002)
9. Brom, C., Gemrot, J., Michal, B., Ondrej, B., Partington, S.J., Bryson, J.J.: Posh tools for game agent development by students and non-programmers. In: *IEEE 9th Computer Games Conference (CGames 2006)* (2006)
10. Brooks, R.: A robust layered control system for a mobile robot. *IEEE J. Robot. Autom.* **2**(1), 14–23 (1986)
11. Brooks, R.A.: Intelligence without representation. *Artif. Intell.* **47**, 139–159 (1991)
12. Bryson, J.J.: Action selection and individuation in agent based modelling. In: Sal-lach, D.L., Macal, C. (eds.) *Proceedings of Agent 2003: Challenges in Social Simulation*, pp. 317–330. Argonne National Laboratory, Argonne (2003)

13. Bryson, J.J., Ando, Y., Lehmann, H.: Agent-based modelling as scientific method: a case study analysing primate social behaviour. *Philos. Trans. Roy. Soc. B Biol. Sci.* **362**, 1685–1699 (2007)
14. Bryson, J.J., Theodorou, A.: How society can maintain human-centric artificial intelligence. In: Toivonen-Noro, M., Saari, E., Melkas, H., Hasu, M. (eds.) *Human-Centered Digitalization and Services*. Springer, Singapore (2019). [https://doi.org/10.1007/978-981-13-7725-9\\_16](https://doi.org/10.1007/978-981-13-7725-9_16)
15. Bryson, J.J.: Intelligence by design: principles of modularity and coordination for engineering complex adaptive agents. Ph.D. thesis, MIT (2001)
16. Castelfranchi, C., Dignum, F., Jonker, C.M., Treur, J.: Deliberative normative agents: Principles and architecture. In: *International Workshop on Agent Theories, Architectures, and Languages* (1999)
17. Dennett, D.C.: The age of post-intelligent design. In: *The Age of Artificial Intelligence: An Exploration* (2020)
18. Dignum, F.: Agents for Games and Simulations II: Trends in Techniques, Concepts and Design. Springer, Heidelberg (2011). <https://doi.org/10.1007/978-3-642-18181-8>
19. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. Ph.D. thesis, SIKS (2004)
20. Dignum, V., Dignum, F.: Agents are dead. Long live agents! In: *19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (2020)
21. Gaudl, S., Davies, S., Bryson, J.J.: Behaviour oriented design for real-time-strategy games: an approach on iterative development for StarCraft AI. In: *Foundations of Digital Games (FDG)* (2013)
22. Guzel, M.S., Bicker, R.: A behaviour-based architecture for mapless navigation using vision. *Int. J. Adv. Robot. Syst.* **9**, 18 (2012)
23. Hannoun, M., Boissier, O., Sichman, J.S., Sayettat, C.: MOISE: an organizational model for multi-agent systems. In: Monard, M.C., Sichman, J.S. (eds.) *IBERAMIA/SBIA 2000*. LNCS (LNAI), vol. 1952, pp. 156–165. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-44399-1\\_17](https://doi.org/10.1007/3-540-44399-1_17)
24. Hübner, J.F., Sichman, J.S.a., Boissier, O.: A model for the structural, functional, and deontic specification of organizations in multiagent systems. In: *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence* (2002)
25. 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. Found. Eng.* **1**, 370–395 (2007)
26. Kahneman, D., Frederick, S.: A model of heuristic judgment. In: *The Cambridge Handbook of Thinking and Reasoning* (2005)
27. Kollingbaum, M.J., Norman, T.J.: Norm adoption and consistency in the NoA agent architecture. In: Dastani, M.M., Dix, J., El Fallah-Seghrouchni, A. (eds.) *ProMAS 2003*. LNCS (LNAI), vol. 3067, pp. 169–186. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-25936-7\\_9](https://doi.org/10.1007/978-3-540-25936-7_9)
28. Lee, J., Padget, J., Logan, B., Dybalova, D., Alechina, N.: N-Jason: Run-time norm compliance in AgentSpeak. In: *International Workshop on Engineering Multi-Agent Systems* (2014)
29. Maes, P.: Situated agents can have goals. *Robot. Auton. Syst.* **6**, 49–70 (1990)
30. Mataric, M.J.: Behaviour-based control: examples from navigation, learning, and group behaviour. *J. Exp. Theor. Artif. Intell.* **9**, 323–336 (1997)

31. Matellán, V., Borrajo, D.: ABC(2) an agenda based multi-agent model for robots control and cooperation. *J. Intell. Robot. Syst.* **32**, 93–114 (2001). <https://doi.org/10.1023/A:1012009429991>
32. Song, I., Governatori, G.: Designing agent chips. In: 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006) (2006)
33. Song, I., Governatori, G., Diederich, J.: Automatic synthesis of reactive agents. In: 11th International Conference on Control, Automation, Robotics and Vision (ICARCV) (2011)
34. Theodorou, A., Bandt-law, B., Bryson, J.J.: The sustainability game: AI technology as an intervention for public understanding of cooperative investment. In: IEEE Conference on Games (2019)
35. Theodorou, A., Wortham, R.H., Bryson, J.J.: Designing and implementing transparency for real time inspection of autonomous robots. *Connect. Sci.* **29**, 230–241 (2017)
36. Wortham, R.H., Gaudl, S.E., Bryson, J.J.: Instinct : a biologically inspired reactive planner for embedded environments. In: ICAPS 2016 PlanRob Workshop (2016)