Designing a Cognitive Agent Connector for Complex Environments: A Case Study with StarCraft

Vincent J. Koeman, Harm J. Griffioen, Danny C. Plenge, and Koen V. Hindriks

Delft University of Technology, The Netherlands
{v.j.koeman,h.j.griffioen,p.c.plenge,k.v.hindriks}@tudelft.nl

Abstract. The evaluation of cognitive agent systems, which have been advocated as the next generation model for engineering complex, distributed systems, requires more benchmark environments that offer more features and involve controlling more units. One issue that needs to be addressed time and again is how to create a connector for interfacing these agents with such richer environments. Cognitive agents use knowledge technologies for representing state, their actions and percepts, and for deciding what to do next. Issues such as choosing the right level of abstraction for percepts and action synchronization make it a challenge to design a cognitive agent connector for more complex environments. The leading principle for our design approach to connectors for cognitive agents is that each unit that can be controlled in an environment is mapped onto a single agent. We design a connector for the real-time strategy (RTS) game StarCraft and use it as a case study for establishing a design method for developing connectors for environments. StarCraft is particularly suitable to this end as it requires the design of complicated strategies for coordinating hundreds of units that need to solve a range of challenges including handling both short-term as well as long-term goals. We draw several lessons from how our design evolved and from the use of our connector by more than 300 students. Our connector is the first to provide full access for cognitive agents to StarCraft.

1 Introduction

Current cognitive agent technology offers a viable and promising alternative to other approaches for engineering complex distributed systems [5, 11]. However, Hindriks [5] also concludes that “if [cognitive] agents are advocated as the next generation model for engineering complex, distributed systems, we should be able to demonstrate the added value of [multi] agent systems.” Designing a connector that can demonstrate this added value by connecting cognitive agents with an environment that puts strict real-time constraints on the responsiveness of agents, requires coordination at different levels (ranging from a few agents to large groups of agents), and requires complex reasoning about long-term goals under a high level of uncertainty is not a trivial task. The connectors that are currently available for use with cognitive agent systems have remained rather simple, and thus do not fully demonstrate the value of cognitive technologies.

In this paper, we aim to establish a *design approach for developing connectors for complex environments*, aimed at facilitating the development of more connectors that can be used to demonstrate the ease of use of cognitive technologies for engineering large-scale complex distributed systems for challenging environments. We believe that RTS games that deploy large numbers of units provide an ideal case study to this end [3, 13]. The basic idea is to control each unit with a cognitive agent. Based on this, and in accordance with Google (DeepMind) and many other AI researchers [10, 12], we believe that StarCraft is the most suitable RTS game to target in our case study. Moreover, several popular competitions exist for StarCraft AI that can serve as a benchmark for implementations that use cognitive technologies [12]. By carefully designing and efficiently implementing a cognitive agent connector to StarCraft, and then testing this connector with large groups of students, we iteratively refine our approach for the development of agent-environment connectors.

Our focus in this paper is on the case study of designing a connector that enables and facilitates the use of cognitive agent technology for engineering strategies for StarCraft (Brood War) based on a *one-to-one unit-agent mapping*, which is different from most existing StarCraft AI implementations. This unit-agent mapping introduces important challenges that need to be addressed:

1. The connector should facilitate a MAS that operates at a level of *abstraction* that is appropriate to cognitive agents.
2. The connector should be sufficiently *performant* in order to support a sufficient variety of viable MAS implementations using cognitive agents (i.e., both different approaches to implementing strategies as well as the use of different agent platforms).

In other words, the connector design should not force a cognitive MAS to operate at the same level of detail as bots written for StarCraft in C++ or Java, but also not promote the other extreme and abstract too much (e.g., clearly the extreme abstraction of providing a single action 'win' is not useful). To make optimal use of the reasoning typically employed by cognitive agents, the connector should leave low-level details to other control layers whilst still allowing agents sufficiently fine grained control.

2 Related Work

Connectors that support connecting cognitive agent technology to games have been made available for other games [2]. So far, however, most connectors have remained rather simple. The most complex cognitive multi-agent connectors that have been made available so far, are connectors for Unreal Tournament [6]. The design of such a connector involves similar issues related to the facilitated level of abstraction and the resulting performance as in this work. However, the resulting implementation as reported on by Hindriks et al. [6] does not support running more than 10 agents, whereas for a StarCraft interface we need to connect hundreds of cognitive agents to control the hundreds of units in game. Moreover, corresponding agent systems for Unreal Tournament generally offer only a very restricted set of actions that agents can perform (i.e., mostly just a “go to” action because other middleware software is used to take care of path planning, shooting, etc.) or communication (i.e., mostly just informing others about enemy positions), limiting the complexity of decision making that is required. Relatively speaking, compared to StarCraft, the diversity in strategies or tactics that can be deployed is rather small. It is therefore not feasible to derive a design approach for connectors to richer environments from this work.

RTS games are widely regarded as an ideal testbed for AI [10, 13]. An RTS game like StarCraft involves long-term high-level planning and decision making, but also short term control and decision-making with individual units. This distinction between respectively strategical and tactical decision making is generally referred to as *macro* and *micro* respectively. These factors and their real-time constraints with hidden information make RTS games like StarCraft ideal for iterative advancement in addressing fundamental AI challenges [13]. Although machine learning solutions have been applied to some problems at the micro level, learning techniques have not been successfully applied to other aspects, mainly due to the vast state spaces involved [12]. The concepts of cognitive agents seem to be a good fit for addressing these challenges, allowing individual cognitive agents to reason about their tactical decision making whilst also inherently facilitating communication to make decisions at a joint strategical level. The reasoning typically applied by cognitive agents seems to lend itself for macro really well, but such systems can potentially employ learning techniques to perform specific sub-tasks (at the micro level) as well.

The prototypical RTS game is StarCraft [12], originally developed by Blizzard in 1998, but still immensely popular both in (professional) gaming and AI research. An API for StarCraft (Brood War) has been developed for several years: BWAPI [4]. BWAPI reveals the visible parts of the game state to AI implementations, facilitating the development of competitive (non-cheating) bots. Several dozens of such bots have been created with this API, mostly written in C++ or Java, aimed at participating in one of the tournaments that are being held for StarCraft AI implementations. However, this work does not directly facilitate cognitive agents that use knowledge technologies and realise a one-to-one unit-agent mapping. A first attempt at creating a cognitive interface for StarCraft was performed by Jensen et al. [7]. In this work, a working proof-of-concept that ties in-game units to cognitive agents was introduced. However, it does not address the major challenges such an implementation faces concerning the level of abstraction and corresponding performance, as we do in this work. When using this connector, it is not possible to create viable (diversities of) strategies, as the range of strategies it supports is quite limited. This connector only offers a small subset of all possible actions associated with each unit in the game, and the percepts made available by the connector do not provide sufficient information for in game decision making either. In this work, we aim to allow virtually any strategy to be implemented with a sufficient level of performance using a cognitive agent connector based on the design approach we propose.

3 Case Study: StarCraft

In StarCraft, each of the three playable races have their own set of unit types. Although many races share similar types of buildings (e.g., depots to bring resources to), there are also substantial differences to take into account (e.g., units 'morphing' into a different type of unit). For most types of units there are usually multiple 'instances' (i.e., individual units) in a game, thus allowing anywhere from 5 up to 400 units representing one army in the game at a certain time. Depending on factors such as game length, the average number of units for one army in a typical game at any point in time is around 100, although many units will also die during the game (i.e., the total number of agents used is much higher). Performance is thus of vital importance, as a substantial performance impact will limit the amount of viable strategies.

Our cognitive agent connector to StarCraft was developed and refined in three iterations. We draw several general lessons from these iterations, which we have incorporated into our proposal for a connector design approach. Initially, a pilot was held with around 100 Computer Science masters students that worked in groups on creating a StarCraft bot using this connector. More recently, over 200 first year Computer Science bachelors students did the same with an improved version of the connector, being the largest StarCraft AI project so far. We continued development of the connector after this project, and made several additional improvements.

4 Connector Design Approach

In this section, we discuss our design approach for a cognitive agent connector. The core of such a connector consists of three components: (i) the *entities* that are provided for agents to connect to (i.e., units in an RTS game), (ii) the outputs that are generated by each entity (and thus which *percepts* a corresponding agent receives), and (iii) the inputs that are available for each entity (and thus which *actions* an agent controlling the entity can perform). We make some basic assumptions about the architecture of a *cognitive agent.* We assume such an agent pro-actively reasons about the *actions* that it should take based on (for example) its goals and beliefs in some fixed *decision cycle* that is asynchronous from the environment in which it operates (for a certain *entity* in that environment), from which it receives information through *percepts.* Multiple agents can work together in one multi-agent system, which is not centrally controlled but does facilitate direct *messaging* between (groups of) agents. Our connector makes use of the Environment Interface Standard [1] for interacting with MAS platforms.

4.1 Micro and Macro Management

In complex environments such as StarCraft, a crucial distinction exists between top-down strategical decision making (macro) and bottom-up tactical decision making (micro). The basic assumption that we make is that a connector needs to provide support for a multi-agent approach based on a one-to-one unit-agent mapping, which inherently facilitates decision making from a bottom-up perspective. At the micro level, *every unit that is active in the environment should be mapped onto an entity* that a cognitive agent can connect to in order to control the behaviour of the unit. For StarCraft, this thus means that any moving or otherwise active unit such as a building will be controlled by a cognitive agent.

Although we initially assumed that the emergent behaviour from these agents would be sufficient to cover the strategical aspects, in practice this was hindered by the high dynamicity of an environment such as StarCraft, for example illustrated by the fact that any unit can be killed at any point in time. To facilitate macro management, we therefore have introduced a new, special kind of entities, so-called *managers*, which are made available by the connector. Managers do not match with unique in-game units, and as such they do not naturally have percepts or actions associated with them. However, as they still need to be informed about the state of the game in order to perform strategical decision making, they instead should have the ability to receive desired global information through percepts as for example indicated in the initialization settings.

4.2 Local and Global Information

The set of available percepts determines what information a specific entity 'sees' during the game, and thus what information its corresponding agent will receive. Percepts have a *name* to describe them and a set of *arguments* that contain the actual data. For example, a percept could be defined as `map(Width, Height)`, and an agent could then receive `map(96, 128)` in a match. In order to determine the percepts that are created for each type of unit, our approach proposes several design guidelines. A key foundation of our approach to handling information from complex environments such as StarCraft is that there is a difference between '*local*' information that is specific to a certain unit in the game (e.g., a unit's health) and '*global*' information that is potentially relevant to all units (e.g., the locations of enemy units). An agent should be able to *perceive all local information* that is specific to its corresponding unit's state, whilst a manager agent should be able to *perceive all global information* that is needed for its strategic (macro) reasoning. However, pieces of global information might also be needed in the agent for a specific unit (e.g., nearby enemy units in StarCraft).

To this end, we initially pushed all global information to all unit and manager entities, as a connector cannot determine which parts of this information a specific agent will need. However, our case study showed that this caused a significant performance impact with larger numbers of units. We have therefore found it useful to provide specific mechanisms to a developer to fine-tune the delivery of global percepts. Through the connector's initialization settings, a list of desired 'global information' (i.e., names of percepts) can be given ("subscribed to") for each unit type. This way, a developer can decide which information is relevant for certain agents, instead of such information being sent to agents at all times. This mechanism can also be used for specifying in more detail which global information a certain manager agent needs to be made aware of. Finally, we assume that when local information is needed for macro reasoning, this can be sent to the appropriate manager agent by the agent for a specific unit within the agent platform; it is thus not required to handle this within the connector.

The ease of use of the percepts for an agent programmer should also be taken into consideration, i.e., by grouping related pieces of information together. The design guideline here is that one should *only group sets of parameters that naturally belong together*. Moreover, to avoid having to deal with different kinds of percepts for each type of unit, a design guideline is that *the percepts should be as generic as possible in order to facilitate re-use* between different agents. This guideline is aimed at reducing the number of different concepts introduced in our percept ontology, and thus aims for efficiency of design.

Performance One of the main challenges is how to deliver all percepts while guaranteeing sufficient performance levels. It is important to manage the percept load of individual agents, as creating the information needed for percepts (i.e., in the connector) and relaying that information to one or multiple agents who then have to make this information available for use in reasoning (i.e., by representing them in a Prolog base) is the most resource intensive task in a connector. In contrast to actions, of which usually at most one is selected per decision cycle, there are usually many percepts (all containing various amounts of information) sent to each agent per decision cycle. We therefore introduce a number of optimization guidelines which aim to either reduce the total number of percepts an agent will have (to store) or the amount of updates to this set of percepts that an agent will have to process.

Complex environments have a lot of static information to which all individual agents may need to have access, like what a certain unit costs to produce or what kinds of units a certain building can produce in StarCraft. Because such environments also introduce many units (and thus many agents), the initialization costs for such information for each of these agents can have a rather big impact on a connector's performance. To avoid this issue as much as possible, we introduce another design guideline to *only create percepts for information that changes in a single match or between matches*. Static information is better suited to be encoded in the agent system itself instead of being sent through percepts, as this will significantly reduce the performance when initializing an agent (which as aforementioned can happen many times during a game as large numbers of units come and go almost constantly). To this end, information that is fixed by the game itself can be coded as a separate part of the ontology that can and needs to be loaded only once at the start of the game. Agents will still need to be informed about changes between matches, e.g., map-specific information should not be included in the 'fixed part' of the ontology. Another guideline to keep the number of percepts low is to ensure that *no data is sent through percepts that can either be calculated based on other data* (e.g., the number of friendly units by counting the number of percepts about their status), *or retrieved from other agents* (e.g., the position of a friendly unit). Relaying information (like friendly unit positions) through messaging between the agents in a MAS is usually much more efficient, as an agent programmer can then selectively choose at which times and to which units to send specific pieces of information, as opposed to percepts always being sent to certain units even when they do not require them (at that time). For StarCraft, combining the (finite set of) information that is available through the BWAPI interface with the guidelines as posed in this section lead to a set of about 25 percepts[1].

5 Conclusions and Future Work

We have presented a design approach for creating connectors for cognitive agent technology to (complex) environments, illustrated by a case study of a cognitive agent connector that provides full access to StarCraft. A major challenge that was addressed during the development of this connector was to ensure corresponding agent systems can be programmed at a high level of abstraction whilst simultaneously allowing sufficient variety in strategies to be implemented by such systems. Based on this challenge, design guidelines for determining the set of available set in agent-environment connectors were determined. The viability of our approach is demonstrated by multiple large-scale practical uses of the StarCraft connector, resulting in a varied set of competitive AIs.

Ensuring a sufficient level of performance of the connector was a significant challenge that had to be addressed in particular in order to demonstrate that a unit-agent mapping (MAS) approach is viable. In our evaluations, which we cannot extensively report on in this paper due to space constraints, we determined the baseline performance of the connector in a worst-case scenario, which shows that on average there remains sufficient CPU time for strategic reasoning in a cognitive MAS. Even though the performance of such a MAS depends largely on the agent technology used itself, we believe that our connector can be effectively used in practice. Although our case study is focused on the 'Brood War' version of StarCraft, the brand new 'raw API' of StarCraft 2 is reported to be similar to BWPAI in Vinyals et al. [14], and the work in this paper should therefore be relatively straightforwardly applicable and/or portable to StarCraft 2 (and possibly other RTS games) in future work.

Finally, through the development and use of our connector for StarCraft, a number of challenges to cognitive agent technologies were identified. One of those challenges is the fact that debugging (cf. Koeman et al. [9]) becomes increasingly difficult with increasing numbers of agents. As debugging concurrent programs is a hard problem in general, more work is required in this area; it could for example be useful to visualize the interaction between agents or the CPU time required by each agent. In addition, in order to better support automated testing, (cf. Koeman et al. [8]), it may be beneficial to develop a mechanism that automatically saves the state of a MAS when a save game is created in StarCraft. This can be used to immediately initialize a MAS to the desired state when executing a test with a specific save game (i.e., a scenario).

[1] For the full set of percepts and actions, we refer to `https://github.com/eishub/Starcraft/blob/master/doc/Resources/StarCraftEnvironmentManual.pdf`.

References

1. Behrens, T.M., Hindriks, K.V., Dix, J.: Towards an environment interface standard for agent platforms. Annals of Mathematics and Artificial Intelligence 61(4), 261–295 (2011)
2. Dignum, F.: Agents for games and simulations. Autonomous Agents and Multi-Agent Systems 24(2), 217–220 (Mar 2012)
3. Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and agents: Designing intelligent gameplay. International Journal of Computer Games Technology 2009 (2009)
4. Heinermann, A.: Brood War API. `https://github.com/bwapi/bwapi` (2008), accessed: 2018-05-12
5. Hindriks, K.V.: The Shaping of the Agent-Oriented Mindset, pp. 1–14. Springer International Publishing (2014)
6. Hindriks, K.V., van Riemsdijk, B., Behrens, T., Korstanje, R., Kraayenbrink, N., Pasman, W., de Rijk, L.: Unreal GOAL bots. In: Dignum, F. (ed.) Agents for Games and Simulations II: Trends in Techniques, Concepts and Design, pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
7. Jensen, A.S., Kaysø-Rørdam, C., Villadsen, J.: Interfacing agents to real-time strategy games. In: SCAI. pp. 68–77 (2015)
8. Koeman, V.J., Hindriks, K.V., Jonker, C.M.: Automating failure detection in cognitive agent programs. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. pp. 1237–1246. AAMAS '16, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2016)
9. Koeman, V.J., Hindriks, K.V., Jonker, C.M.: Designing a source-level debugger for cognitive agent programs. Autonomous Agents and Multi-Agent Systems 31(5), 941–970 (Sep 2017)
10. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.: A review of computational intelligence in RTS games. In: 2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI). pp. 114–121 (Apr 2013)
11. Logan, B.: A Future for Agent Programming, pp. 3–17. Springer International Publishing, Cham (2015)
12. Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game AI research and competition in StarCraft. IEEE Transactions on Computational Intelligence and AI in Games 5(4), 293–311 (Dec 2013)
13. Robertson, G., Watson, I.: A review of real-time strategy game AI. AI Magazine 35(4), 75–104 (2014)
14. Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A.S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al.: StarCraft II: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 (Aug 2017)