

# CIGA: A Middleware for Intelligent Agents in Virtual Environments

Joost van Oijen<sup>1,2</sup>, Loïs Vanhée<sup>3,1</sup>, and Frank Dignum<sup>1</sup>

<sup>1</sup> University of Utrecht

PO Box 80.089, 3508 TB Utrecht, the Netherlands

{oijen,lois,dignum}@cs.uu.nl

<sup>2</sup> VSTEP

Weena 598, 3012 CN Rotterdam, the Netherlands

joost@vstep.nl

<sup>3</sup> ENS de Cachan - Antenne de Bretagne

**Abstract.** Building intelligent behavior in (educational) games and simulations can greatly benefit from the use of agent technology. Intelligent agents within a multi-agent system can be developed for controlling virtual characters in a simulation environment within a game engine. Coupling a multi-agent system to a game engine is not a trivial task and introduces several conceptual design issues concerning embodied agent design. In this paper we present CIGA, a middleware to facilitate this coupling tackling the design issues in a structured approach, not only for embodied agent design but also for the system as a whole.

**Key words:** Middleware, Multi-Agent Systems, Virtual Environments, Intelligent Agents, Simulation

## 1 Introduction

As the technology to create more realistic, complex and dynamic virtual environments advances, there is an increasing interest to create intelligent virtual agents (IVAs) to populate these environments for the purpose of games, simulations or training. Designing an IVA, game engine technology can be employed to simulate its physical embodiment, equipped with sensors and actuators interacting with the virtual environment. The use of agent technology in the form of multi-agent systems (MASs) is a good fit to realize the cognitive and decision-making aspects of an IVA.

Combining these technologies is not a trivial task and introduces conceptual and technical design issues. First of all, both technologies often work at different abstraction levels. Games engines work with low-level data representations for virtual environments and the characters populating it. MASs work with more high-level semantic concepts designed to form a suitable abstraction from the physical environment representing an agent's perceptive view on the environment and the actions for influencing it. Second, agent actions in a typical MAS

environment are non-durative. When embodied in a real-time environment, actions become durative and low-level reasoning over their execution is required. MASs are generally not designed to handle this aspect. Third, the designs of an agent's embodiment in a game engine and its cognitive counterpart in the MAS are highly dependent on each other. An agent's view on the environment is dependent on his sensory capabilities provided by its embodiment whereas its ability to influence the environment through its embodiment is bounded by the possible control over an avatar in the game engine. This, in turn, has implications on an agent's deliberation on possible goals, plans and actions. Last, the required connection between the two specialized software systems introduces some technical issues concerning software engineering.

Current attempts in combining these technologies often use a pragmatic approach when tackling these design issues. A direct connection between a game engine and a MAS is created either with or without the help of standard technologies or interfaces that may provide access to a specific game engine [1] or range of MASs [3]. Although such an approach can be a productive solution, design decisions are often influenced and bounded by the individual capabilities of the employed technologies. There is often no structured approach in bridging the conceptual gap between the two systems. There are systems that focus more on the conceptual issues attempting to employ agent technology by translating the physical world model to a social world model suitable for cognitive reasoning [5, 16]. Though, these systems don't pretend to give a structured approach for tackling the technical issues when using their system with alternative MASs or game engines.

In this paper we present CIGA<sup>1</sup>, a middleware to facilitate the coupling between agent technology and game engine technology, tackling the inherent design issues in a structured way. An architecture for using this middleware is presented to solve technical issues when connecting agents in a MAS to their embodiments in a game engine. Additionally we show the need for using ontologies to provide a design contract between not only an agent's mind and body connection but also for the system as a whole. This can lead to a new methodology for game design using agent technology. An initial fully functional version of CIGA has been implemented and currently undergoes evaluation.

The paper is organized as follows. In the following section we outline the motivation for introducing CIGA. Section 3 describes the architectural design of the proposed middleware. In section 4 we compare the middleware with related technologies. Finally, section 5 we conclude and discuss results.

## 2 Bridging the Conceptual Gap

Designing an embodied agent with current game engine and agent technology, one must overcome several inherent conceptual design issues. In this section, we provide a description of each issue and present the functional role CIGA plays to overcome these issues.

---

<sup>1</sup> Creating Intelligent Games with Agents

## 2.1 Social World Model

In a MAS, an agent’s interpretation of the environment is based on semantic concepts forming an abstraction of the virtual physical world. The data representations of these concepts in a game engine often are at a different abstraction level than what is suitable for agents in a MAS to work with. For example, an agent’s concept of ”a person sitting on a chair” may be represented in the game engine by a character’s location in the vicinity of a chair in combination with the positions of each skeletal bone, forming a sitting posture. Instead of the physical world state representations in a game engine, agents work with (high-level) semantics concepts. The use of rich semantic concepts is particularly important for the more socially-oriented simulations with communicating agents like in serious games. Though, the demand for rich semantics in the more action-oriented games is getting increasingly important [17].

CIGA overcomes the difference of data representation by translating the *physical simulation* to a *social simulation* for agents. To accomplish this, semantic data is generated during agent sensing which is translated from raw data of game objects or events. This semantic information forms the basis for an agent’s view and interpretation of the environment. Inferences can be made to provide agents with semantic concepts relevant for the social simulation. For example, the meaning of a certain gesture performed by an embodied agent can be inferred from an animation in the game engine. Making higher-level information directly available for agents is efficient as agents don’t have to infer these themselves. CIGA employs *domain ontologies* to specify a formal representation of the semantic concepts. The ontologies are accessible in both the game engine to perform the required translations and in the MAS representing the agent’s *social world model*.

The risk of translating raw data to semantic data is the problem of *over-inference*. As implemented inferences are the same for all agents we might make an inference we don’t want a certain agent to be able to make. Further, we might lose the ability for an agent to interpret perceived information in his own way. For example, how an agent interprets the meaning of a perceived gesture can be dependent on an agent’s cultural identity. This makes it important to design the semantic concepts in the domain ontology at the right abstraction level such that agents don’t have to perform too much low-level inferences on their own, but can still make different interpretations based on their individual context. CIGA doesn’t enforce the use of any abstraction level as this is dependent on the application domain.

## 2.2 Perception

Agents in a MAS get information from their environment through percepts. If an agent becomes embodied in a virtual environment, these percepts are based on sensory information retrieved from one or more sensors attached to the embodiment in a game engine. When creating percepts directly from sensory information, we do not only face the problem of information representation as described

above, but there's also the risk for an agent to become flooded with percepts that are irrelevant with respect to his current state of mind. An agent should have the ability to direct his attention to selected information from the environment such that irrelevant information can be filtered, though still allowing an agent to be susceptible to unexpected events. Filtering sensory information should not be performed in the game engine as this process is dependent on the agent's mind in a MAS. On the other hand, delegating this process to the MAS is not ideal as the cost of communicating the unfiltered information can have a negative performance impact on the system as a whole. Additionally, MASs generally don't provide standard facilities implementing perception filtering.

CIGA tackles this problem by introducing a *filtering mechanism* located closely to an agent's sensors in the game engine. Agents in a MAS can show their *interest* in the environment in the form of subscriptions that define how sensory information has to be filtered. Using the environment semantics defined in the domain ontology introduced before, powerful subscriptions can be made to give an agent full control over the range of percepts to receive. A description of this mechanism can be found in [20] and falls out of the scope of this paper.

### 2.3 Action

In a MAS, an agent's capability to influence the environment is defined by a set of actions designed to change the state of an environment. The success or failure of an action denotes that the desired state of the environment was reached or could not be reached respectively. In a typical MAS environment, actions are instantaneous and the result is known immediately. When an agent becomes embodied in a virtual environment, its capability to influence the environment becomes bounded by the available actuators of the embodiment. Since these actuators work in real-time, actions become durative and the environment may change during the execution, possibly preventing the action from finishing successfully. For example, an action like *open door* can fail during execution if the door is opened from the other side by another agent. Further, this raises questions about the meaning of the success or failure of an action performed by an embodied agent. Is an action said to be successfully executed if the body performed the action or if the desired state of the environment was reached?

To deal with this different view on *actions*, CIGA provides a generic *action monitoring* facility to deliver action requests from an agent to its embodiment and communicate feedback about the realization of the action, allowing an agent to follow the progress and intervene if necessary. The meaning of the success or failure of an action is left to the designer where he can use the feedback mechanism to specify the state of the action and how it was reached (E.g. the agent didn't fully perform the action in the environment, but we still consider the action to be succeeded as the desired environment state defined by the semantics for the action was reached). A more elaborate overview is given in section 3.2.

A well known design issue is the need for finding a suitable abstraction level for behavior control. Choosing an abstraction level has implications on both

agent design and system performance. The use of more low-level, physically-oriented actions gives an agent more control over its body but increases the communication cost of delivering the instructions to the game engine. Using more high-level, cognitive-oriented actions delegates more control to the game engine, but the ability is lost to take an agent’s individuality into account. For example, it becomes harder to reflect an agent’s own personality or mental state on his behavior if this information is defined in the MAS (E.g. drunk and sober agents will walk in the same way). Although the communication cost for sending instructions is decreased, agents are more dependent on perception to see if the intents of their actions have been achieved. The aim is to find the right *balance of intelligence* distributed between the mind and body of an agent in the MAS and game engine respectively. CIGA doesn’t enforce the use of any abstraction level for actions as this is dependent on the specific application domain.

## 2.4 Communication

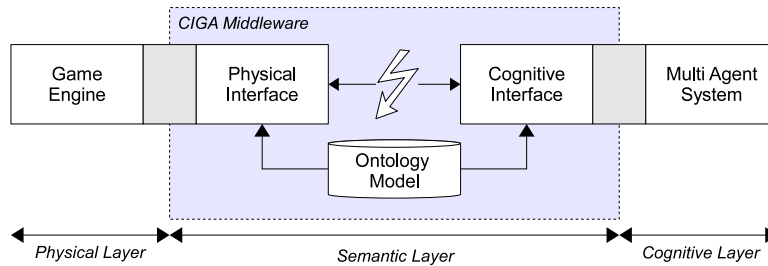
MASs often provide an inter-agent communication mechanism for agents to communicate. The messages being communicated usually adhere to standards like FIPA ACL where content can be represented using formal semantics understood by both agents. Simulating human-like communication requires agents to perform (non)verbal communicative behavior and perception through their body’s actuators and sensors in the environment. Like actions, communication becomes durative. Further, the desired effect of the communication cannot easily be determined as this is dependent on mental processes within the receiving agent. Successful reception of a communicative act is not trivial as this depends on the available medium from sender to receiver, bounded by the simulated laws of physics in the environment. For example, two agents may not be able to hear each other in a noisy bar when they are at different sides of the room.

CIGA facilitates in the communication process between embodied agents by introducing its own communication mechanism taking into account both the durative nature of communication and environmental factors. For example, the delivery of a communication message is only performed when the corresponding action realizing the communicative act in the environment is successfully achieved and the receiving agent is physically able to perceive this act. This mechanism is briefly mentioned later in this section but an elaborate functional overview falls outside the scope of this paper.

## 3 CIGA Framework

In this section we present an architectural framework for integrating the CIGA middleware with both game engine and agent technology. An illustration of the main framework is given in figure 1.

Since the proposed middleware must connect to two specialized software systems, the common design approach was taken to internally divide the middleware into two functional components. The *Physical Interface* layer connects to a game



**Fig. 1.** Middleware Framework.

engine whereas the *Cognitive Interface* layer connects to a MAS. Both components are internally connected using a communication mechanism. The *Ontology Model* provides access to domain ontologies specified for a specific application domain containing formal representations of the communicated content between an agent's mind and body.

This internal distributed design several advantages. First, it helps to bridge the conceptual gap between a game engine and a MAS by dedicating separate components for the integration with the technologies. Second, from a technical point of view, it allows both components to be implemented in different programming languages. It is often the case that the used game engine and MAS are written in different languages. For the middleware as a whole to be able to interface with both technologies while matching the language of that technology results in an easy integration process and an efficient, tight connection. Last, the design introduces connection transparency since the game engine and MAS can run in different processes or distributed over different computers or platforms, depending on the used internal connection mechanism.

Next we'll first describe the role of ontologies within the middleware after which we'll look at the individual components connecting to game engine and MAS respectively.

### 3.1 The Role of Ontologies

The *Ontology Model* represents a storage facility for semantic concepts. It consists of domain ontologies designed for a specific application domain to capture an agent's perceptual and interactional capabilities within an environment. The use of ontologies forces an agreement between a game engine and a MAS on the required domain concepts. This is known as a *design by contract* [14], increasing robustness and reusability within the system.

Building a domain ontology for the simulation environment encompasses defining object and event classes with their attributes. Attributes for objects represent their physical or functional properties whereas attributes for events represent parameters specifying event details. Classes can be organized in a hierarchical fashion where attributes are inherited from parent classes. To form an

agreement on the actions agents can perform in an environment, (parameterized) action classes should be specified in the domain ontology.

Domain ontologies can be created using an ontology editor like Protégé. An interesting feature is the ability to change and extend meta-classes for objects, events or actions. This allows the ontology to support custom data fields for specific types of concepts. For example, a *perceptibility type* can be assigned to an object property to specify its perceivability (e.g. visual, auditory or tactile) which use will be described later. Additionally, *affordances* can be specified for object classes which can facilitate agents in understanding their world in terms of interactions they can have with it. The use of Affordance Theory has been previously explored in [6, 4]. Related to affordances, information associated with *smart objects* can be stored [11, 15]. A small example showing the possibilities of a simple domain ontology is illustrated in figure 2.

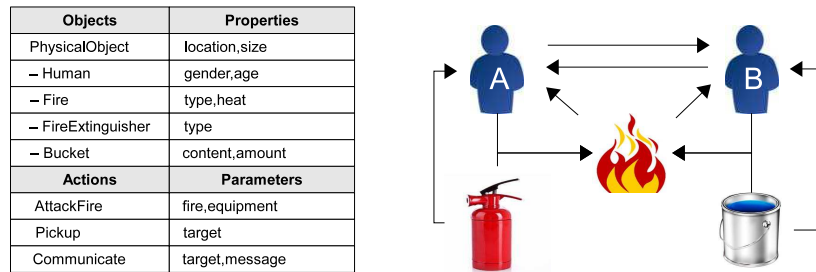


Fig. 2. Domain Ontology Example.

The left side of the illustration shows a domain ontology consisting of several object classes and actions for human actors. The right side of the example shows a simple scene with two human agents (A and B), a fire and two objects which can be used to attack a fire. Now assume *agent A* notices a small fire starting near *agent B* who is unaware of this. *Agent A* would like to resolve the situation and has several options. He can pick up the fire extinguisher and use it to put out the fire or he can use the bucket of water positioned near *agent B*. This choice may depend on the size and type of the fire and the type fire extinguisher. For example, an electrical fire should not be extinguished using water which makes the first option preferable if it concerns a chemical fire extinguisher. Further, *agent A* can communicate with *agent B* to advise him to deal with the fire, although this choice may not be suitable if *agent B* is a small child.

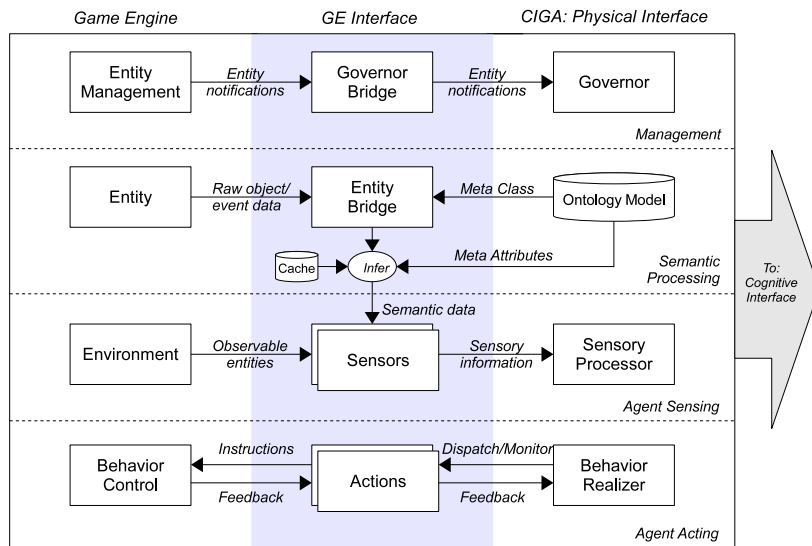
*Agent A* is able to perform this line of reasoning based on the given domain ontology. Here, object properties provide information about the objects (E.g. object positions, the type of fire extinguisher or the age of *Agent B*). Object classes can be annotated with *conditional affordances* helping an agent to understand how he can interact with objects being perceived. For example, physical objects of a certain size can be picked up or a bucket filled with water can be used to

attack a fire. The concepts in the domain ontology are also interesting for use as content in communication languages between agents [21].

These domain ontologies cover concepts relevant to both an agent’s embodiment and his mind. In CIGA they are fully accessible at runtime to both the game engine and MAS as will become clear in the next parts of this section.

### 3.2 Connecting the Game Engine

As shown in Figure 1, the *Physical Interface* layer of CIGA connects to a game engine. Its main task is the administration of agent embodiments participating in the middleware and individually control their sensors and actuators. Abstraction from the game engine is achieved using an intermediate layer, hereafter called the *GE Interface* layer, connecting a specific game engine. Figure 3 illustrates the design focusing on the functional interfaces and data flows.



**Fig. 3.** Integration Middleware in Game Engine.

A prerequisite for using CIGA is the ability to modify the game engine allowing implementation of the *GE Interface* layer. This layer is responsible for integrating CIGA’s *Physical Interface* component as an external game engine component to be included in the engine’s update loop which allows it to run processes on its own. For example, agent sensing can be controlled to run at a configurable frequency or the MAS can be provided with regular time updates. This approach makes CIGA less dependent on specific features that may or may not be available in a specific game engine. Next we describe each horizontal layer from figure 3 in more detail.

**Management** The role of the *Governor* in the *Physical Interface* is to provide a connection mechanism for synchronizing the simulation between the game engine and the MAS. It monitors the creation and destruction of entities in the environment that are candidates for agent embodiment and notifies the MAS about their existence. Additionally, simulation time is synchronized by sending regular time updates to the MAS, who often don't have an internal clock explicitly defined.

**Semantic Processing** The goal of semantic processing is to translate raw object and event data available in the game engine to semantic data. Semantic data is used as sensory information which allows an agent to build a social model of the environment based on meaningful concepts. The *Ontology Model* can be accessed to retrieve the formal representation of those concepts.

Creating semantic data from raw data at runtime is a process performed in the *GE Interface* layer. Here, at design time, entity bridges are created associating object classes from a domain ontology to entities defined in the game engine (E.g. associating the *fire* concept to a fire class in the game engine). During runtime, the object's attributes are generated from raw entity data (E.g. an object's *size* property is calculated). This translation process is performed when agents sense environment entities. Furthermore, semantic event classes can be generated based on raw game events or as a result of custom inferences. Inference based on previously sensed information is achieved using a cache belonging to an agent's sensory processor.

Note that translation and inference rules for generating semantic data are the same for all agents. At this stage, although agents have their individual view of the environment determined by their sensors, their interpretation of it is the same as specified by the domain ontology. This fact must be taken into account when designing the ontology.

**Agent Sensing** Agent sensing is performed using a *Sensory Processor* provided for each participating agent within CIGA. Its goal is to collect sensory information from all sensors assigned to an agent's embodiment and prepare them as *percepts* for the MAS agent to receive.

*Sensors* obtain sensory information from the environment. The processing logic for a sensor is implemented in the *Physical Interface* using a sensor base class. Specific sensors must be created in the *GE Interface* layer and are required to assign a *perceptibility type* (e.g. visual, auditory, tactile) to the sensor and provide an implementation of the abstract method **GetObservableEntities()**. This method is responsible for building a list of entities from the environment the sensor is currently able to observe. Access to game engine queries can support this process (E.g. to determine if an agent can observe another agent standing behind a wall or if a sound can still be heard at a certain distance from its origin). We assume the game engine offers us the functionality to achieve the required queries. With this approach, one can easily build a sensor library in the *GE Interface* layer to store different sensors with more or less advanced algorithms.

Since sensors can be dynamically replaced, one can support different level of details (LODs) for sensors.

Based on the list of *observable entities*, sensing in the base class continues by extracting sensory information from these entities using the *Semantic Processing* described before. The sensor's *perceptability type* is used to filter the object properties and events that can be sensed. E.g., In the example from figure 2, the *heat* property of the *fire* entity can only be sensed by a *tactile* sensor.

After all sensors have been processed, the *Sensory Processor* filters the collected data as further described in [20].

**Agent Acting** Agent behavior is performed using a *Behavior Realizer* provided for each agent participating within CIGA. Its goal is to realize semantic actions instructed by a MAS by managing an action's life cycle and communicating feedback about its state back to the MAS agent. Actions are executed in parallel in an interleaved fashion driven by the game engine loop.

*Actions* themselves are implemented in the *GE Interface* layer. They are responsible for realizing the intended action semantics by accessing game engine instructions. The *Physical Interface* layer provides an abstract base class for actions. Creating specific actions involves implementing the following methods:

- **CheckPreconditions():** This method is called before the action is executed. Here any preconditions can be checked which must pass before the realization of the action can be started. If the preconditions are not met the action will not continue further and the agent is notified.
- **Body():** This is the main execution loop. Here game engine functionality can be addressed to realize the intent of the action. This includes controlling the actuators of the agent's embodiment and monitoring its progress. For virtual characters, this often involves interacting with an *Animation System* in the game engine. The action can end prematurely when problems arise during realization after which the agent is notified about the cause.
- **CheckEffects():** This method is called after the action was successfully realized. Here the intended effects of the action on the game state can be validated. If the effects are not met the action will end with a corresponding notification.
- **OnAborted():** A MAS agent has the ability to abort any scheduled action. This method is called when it decides to do so. Here logic can be implemented to properly interrupt and clean up the action's realization in the game engine.

Note that it is up to the MAS agent to infer success or type of failure of an action based on the received action feedback notifications. Further, CIGA doesn't impose any rules for the implementation technique or data formats used for actions. It merely provides a generic facility to deliver instructions from a MAS agent to its embodiment and to communicate feedback about the realization of these instructions. For example, a common technique for behavior control is the use of *parameterized actions* representing an API for agents to control

their embodiment [1, 22, 2]. In CIGA, *parameterized actions* can be defined in the domain ontology to form an agreement on the used API.

This does not restrict the use of more specialized techniques. For example, upcoming language standards such as *BML* can still be used [12], which is an XML-based language for communicative behavior realization. Here, a single action can be defined for communicative behavior sending BML data and feedback information between the MAS and the game engine.

### 3.3 Connecting the MAS

The *Cognitive Interface* layer of CIGA connects to a MAS, providing a generic interface for agents in a MAS with their embodiment in a real-time environment. This interface should allow for the communication of percept data and action instructions whose data is associated with semantic concepts from the domain ontology. Similar to the interface with the game engine, abstraction from the MAS is achieved using an intermediate layer, hereafter called the *MAS Interface* layer. Figure 4 illustrates the connection framework.

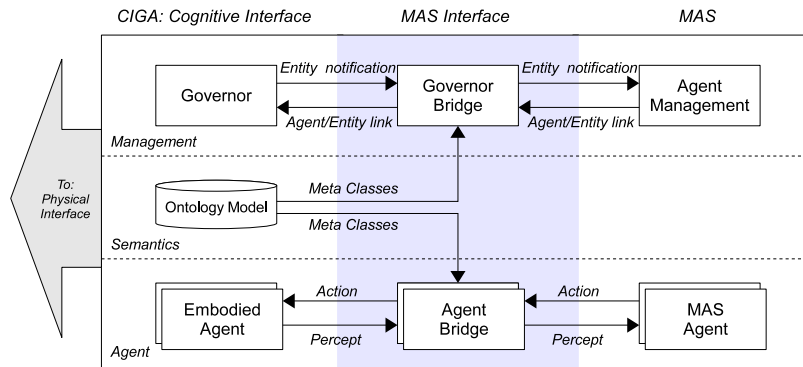


Fig. 4. Integration Middleware in MAS.

Unlike the *Physical Interface*, the *Cognitive Interface* is a pure event-based component passing information to and from the MAS. An *MAS Interface* layer must be implemented for a specific MAS to comply with the provided interfaces by the *Cognitive Interface* layer. This layer is less complex than the *GE Interface* layer since it's a simple message-passing connection for data that is already rooted in semantics (no conceptual translation is required).

**Management Interface** As described previously, the *Governor* notifies the MAS about the creation and destruction of candidate embodiments in the simulation. Based on this information, the MAS can create and destroy agents. To link an agent with an embodiment, the MAS must notify the *Governor* about

the entity it wants to embody. The *Governor Bridge* in the *MAS Interface* can achieve this functionality for a specific MAS to be used. The *Ontology Model* can be accessed to retrieve semantic data about the embodiments. This information can support the MAS in deciding what type of agent to associate with an entity.

**Agent Interface** The agent interface between CIGA and a MAS consists of the common act and sense interfaces required for MAS agents. The *Agent Bridge* in the *MAS Interface* is responsible for converting the different message formats used between the CIGA middleware and a specific MAS. Here, the *Ontology Model* can play several roles. The model can be accessed to retrieve semantic meta-data associated with incoming percepts. For example, agents can retrieve the *affordances* associated with perceived objects. Also type hierarchies of objects in the ontology can be inspected, allowing agents to make generalizations about objects they perceive. In addition, the model can be used to validate the semantics of action instructions performed by a MAS agent. Being able to validate actions can greatly support the development of agents whose code for action-selection cannot be type-checked at design time (E.g. in 2APL).

Three types of percepts have been defined in CIGA's *Cognitive Interface* layer:

- *Object percepts* contain semantic data about objects perceived from the environment. A unique object identifier is provided giving agents the ability to relate subsequent percepts with the same object.
- *Event percepts* contain semantic data about events from the environment. An object's identifier provides the source where the event originated from.
- *Action percepts* contain feedback information about ongoing actions. The MAS agent can associate this feedback with a dispatched action using the included unique action identifier. Feedback information includes the progress status of the action and possible failure conditions.

Two types of actions have been defined in CIGA's *Cognitive Interface* layer:

- *Action instructions* are used for the physical (durative) actions agents perform. They correspond to the actions implemented in the *GE Interface* layer described previously and are executed by the *Behavior Realizer*.
- *Communication instructions* are used for physical communication between agents. These are *special* actions consisting of two parts. The first part contains the physical action the agent performs to realize the communication, corresponding to the previous type of instruction. The second part includes the communicative intent which may be represented in an agent communication language. This part cannot be sent directly to the receiving agent if the physical communication action has not started yet. The *Cognitive Interface* layer is responsible for orchestrating this process.

For the implementation of the *MAS Interface* layer, interface standards like EIS [3] can be employed which has been explored to interface with multi-agent platforms like 2APL, GOAL, Jadex and Jason. Though, such platforms focus

on high-level decision-making and deliberation aspects of agents and lack other aspects of behavior that may be required to form a fully cognitive architecture (E.g. the modeling of physiology, emotion or reflexive behaviors). These aspects can play an important role in simulating virtual humans for example. This issue has been addressed before as seen in CoJACK [8] which extends the JACK platform by combining its symbolic decision-making module with what is called a moderator layer for emotional and physiological factors. The *MAS Interface* layer can easily be used to connect such an additional MAS layer with its environment.

## 4 Related Technologies

In this section, we compare CIGA with related research and technologies with similar functionalities. First we'll look at technologies providing an interface to an environment in a game engine for external access. Gamebots [1] is a modification of the UT game engine and provides fixed sense-act interfaces for in-game avatars accessible using socket communication. It is often used in research on embodied agents mainly because of the lack of good alternatives for accessing virtual environments [7, 9]. Gamebots can be compared to CIGA's *GE Interface* layer (see Figure 3). Though, in Gamebots, there is no methodology for using domain ontologies as the interface messages are fixed and geared specifically towards the UT engine. Further, *action monitoring* is not supported since Gamebots doesn't offer explicit execution and monitoring of actions.

The High Level Architecture (HLA) is an architecture for distributed simulations. Its goal is to synchronize environments running in separate simulations. There have been attempts to connect external agents to simulation environments using HLA [13]. We consider HLA not suitable for connecting MASs since it was not designed for this purpose and therefore lacks facilities for agent-centric sensing and acting. Similarities between CIGA and HLA are the use of *ontologies* as a design contract and the use of a subscription mechanism to control the flow of information send between components. For CIGA, this is described in [20].

Next we'll compare the system of *Pogamut* which has a goal similar to CIGA. *Pogamut* is designed as a mediation-layer between a game engine (GE) and a decision-making system (DMS) to bridge the "representational gap" [10]. It is based on a general abstract framework for connecting a DMS to a GE [9]. The architecture of a *Pogamut* agent consists of a *WorldView* component for GE facts, augmented with optional components like a *Working Memory*, an *Inference Engine*, a *Reactive Layer* and a *DMS*. The main conceptual difference between *Pogamut* and CIGA is that where *Pogamut* presents an agent architecture connected to a game engine, the CIGA middleware offers facilities to connect an agent in a MAS directly to an avatar in the game engine. It doesn't enforce any agent architecture as we consider this to be contained in the MAS. Providing a tight connection with an avatar in the game engine requires CIGA to enforce modifying the game engine. Although this is a strong requirement, we think it is a necessity to better aid in the connection design of an agent's mind and body, allowing us to perform *perception filtering* and *action monitoring* in the

game engine's native programming language. Although Pogamut is more flexible in connecting to different game engines (using Gamebots or HLA), it is highly dependent on the specific game engine. Here, the game engine not only dictates the mechanisms for sensing and acting, but also the use of fixed data representations for actions and sensory information. Although ontologies can be implicitly defined as Java classes, there is no explicit formal agreement between the GE and a Pogamut agent.

Facilitating the connection between MAS agents and MAS environments, the *Environment Interface Standard* (EIS) has been proposed. It provides a general purpose interface for associating environment entities with MAS agents and their sense-act interface [3]. The proposed interface is not primarily geared towards connecting agents directly to a real-time virtual environment. Although EIS can be used for real-time environments, little is said on how to deal with the design issues presented in section 2. EIS has been used in connecting agents to an environment using Pogamut [18].

Last, there are systems which have addressed a subset of the design issues presented in the paper. For example, in [22,16], Mimesis is presented as an architecture to integrate special-purpose intelligent components with a game engine. The architecture addresses both the gap of information representation and action execution, though its design is less geared towards an agent-body connection such that issues in perception and communication are not addressed. In [5], a cognitive middleware layer is introduced which has a similar goal to the semantic processing in CIGA, providing agents with a *social world model*. Unlike CIGA, this system doesn't discuss the technological issues in creating embodied agents. In [19], the *ION Framework* is said to separate the simulation environment from a realization engine. Although it recognizes similar issues, it is unclear about the methods for implementing these guidelines.

## 5 Conclusion & Future Work

In this paper we presented CIGA, a middleware for facilitating the coupling of MASs to game engines providing a connection between a MAS agent and its embodiment in a virtual environment. It is designed as a general-purpose middleware employable in a wide range of applications with different requirements for agents. For example, in one simulation, believable embodied conversational agents (ECAs) are required where detailed (non)verbal communicative behavior and perception is important. In another simulation an agent's interaction and understanding of the environment may be more important requiring a more extended model of the environment and the actions for influencing it. A combination of such qualitative and quantitative aspects may also be desired. Here, CIGA facilitates the development of such simulations by supporting developers to bridge the conceptual gap between a MAS and a game engine without enforcing agent design decisions.

CIGA employs domain ontologies to form an agreement between the game engine and the MAS on the semantics of an agent's perceptual and behavioral

interfaces. This allows designers to formally specify the concepts used within a specific application domain and reference them directly from within the game engine or the MAS. A sensory processing mechanism allows an agent to perceive its environment and build a *social world model* based on formal semantics. Designers are able to choose the required realism for sensors and control the way sensory information is filtered [20]. An action monitoring mechanism enables agents to be synchronized with the realization of their actions performed by their embodiments. Designers are left to provide an implementation of the actions specified for the application domain.

CIGA has been implemented connecting several MASs to an in-house developed game engine<sup>1</sup>. The *Physical Interface* of CIGA has been developed in C++ and the *Cognitive Interface* in Java. The internal connection mechanism employs TCP/IP sockets. MASs that have been tested include *2APL*, *Jadex* and a custom developed MAS testing industry-standard techniques. On top of the middleware platform a graphical user interface has been developed to provide logging and debugging facilities during the development process.

Future work involves validating the principled approach taken by CIGA by exploring different application settings where agents have different requirements. This also involves creating an interface with an alternate game engine. On the conceptual side, further research will be performed concerning the topic of agent communication within CIGA, dealing with formal agent communication in MASs on one side and believable human-like interactions in real-time environments on the other side.

## References

1. R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. Gamebots: A 3d virtual world test-bed for multi-agent research. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001.
2. N. I. Badler, R. Bindiganavale, J. Allbeck, W. Schuler, L. Zhao, and M. Palmer. Parameterized action representation for virtual human agents. In *Embodied Conversational Agents*, pages 256–284. MIT Press, Cambridge, MA, USA, 2000.
3. T. Behrens, K. Hindriks, and J. Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, pages 1–35, 2010.
4. C. Brom, J. Lukavský, O. Šerý, T. Poch, and P. Šafrata. Affordances and level-of-detail AI for virtual humans. In *Proceedings of Game Set and Match 2*, 2006.
5. P. H.-M. Chang, K.-T. Chen, Y.-H. Chien, E. Kao, and V.-W. Soo. From reality to mind: A cognitive middle layer of environment concepts for believable agents. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 57–73. Springer Berlin / Heidelberg, 2005.
6. J. Cornwell, K. O’Brien, B. Silverman, and J. Toth. Affordance theory for improving the rapid generation, composability, and reusability of synthetic agents and objects. In *Proceedings of the Twelfth Conference on Behavior Representation in Modeling and Simulation*, 2003.

---

<sup>1</sup> [www.vstep.nl](http://www.vstep.nl)

7. N. P. Davies, Q. Mehdi, and N. Gough. A framework for implementing deliberative agents in computer games. In *Proceedings of the 20th European Conference on Modeling and Simulation (ECMS'06)*, 2006.
8. R. Evertsz, F. E. Ritter, P. Busetta, M. Pedrotti, and J. L. Bittner. CoJACK - Achieving Principled Behaviour Variation in a Moderated Cognitive Architecture. In *Proceedings of the 17th Conference on Behavior Representation in Modeling and Simulation*, 2008.
9. J. Gemrot, C. Brom, and T. Plch. A periphery of Pogamut: from bots to agents and back again. In F. Dignum, editor, *Agents for Games and Simulations II*, Lecture Notes in Computer Science, pages 19–37, Berlin, Heidelberg, 2011. Springer-Verlag.
10. J. Gemrot, R. Kadlec, M. Bída, O. Burkert, R. Píbil, J. Havlíček, L. Zemčák, J. Šimlovič, R. Vansa, M. Štolba, T. Plch, and C. Brom. Pogamut 3 can assist developers in building ai (not only) for their videogame agents. In F. Dignum, J. Bradshaw, B. Silverman, and W. Doesburg, editors, *Agents for Games and Simulations*, pages 1–15. Springer-Verlag, Berlin, Heidelberg, 2009.
11. M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. In *Proc. Eurographics Workshop on Animation and Simulation*, pages 73–86, 1998.
12. B. Krenn, S. Marsella, A. N. Marshall, H. Pirker, K. R. Thórisson, and H. Vilhjálmsson. Towards a Common Framework for Multimodal Generation in ECAs: The Behavior Markup Language. In *In Proceedings of the 6th International Conference on Intelligent Virtual Agents, Marina del Rey*, 2006.
13. M. Lees, B. Logan, and G. Theodoropoulos. Agents, games and hla. *Simulation Modelling Practice and Theory*, 14(6):752–767, 2006.
14. B. Meyer. Applying "design by contract". *Computer*, 25:40–51, October 1992.
15. C. Peters, S. Dobbyn, B. MacNamee, and C. O'Sullivan. Smart objects for attentive agents. In *WSCG*, 2003.
16. M. O. Riedl. Towards Integrating AI Story Controllers and Game Engines: Reconciling World State Representations. In *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation and Learning in Computer Games*, 2005.
17. T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. Kraker. The role of semantics in games and simulations. *Computers in Entertainment*, 6:57:1–57:35, 2008.
18. K. v Hindriks, B. van Riemsdijk, T. Behrens, R. Korstanje, N. Kraayenbrink, W. Pasman, and L. de Rijk. Unreal GOAL Bots. In F. Dignum, editor, *Agents for Games and Simulations II*, Lecture Notes in Computer Science, pages 1–18. Springer-Verlag, Berlin, Heidelberg, 2011.
19. M. Vala, G. Raimundo, P. Sequeira, P. Cuba, R. Prada, C. Martinho, and A. Paiva. ION Framework A Simulation Environment for Worlds with Virtual Agents. In Z. Ruttkay, M. Kipp, A. Nijholt, and H. Vilhjálmsson, editors, *Intelligent Virtual Agents*, volume 5773 of *Lecture Notes in Computer Science*, pages 418–424. Springer Berlin / Heidelberg, 2009.
20. J. van Oijen and F. Dignum. A Perception Framework for Intelligent Characters in Serious Games. In *In Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011) to appear*, 2011.
21. J. van Oijen, F. Dignum, and W. van Doesburg. Goal-Based Communication Using BDI Agents as Virtual Humans in Training: An Ontology Driven Dialogue System. In F. Dignum, editor, *Agents for Games and Simulations II*, Lecture Notes in Computer Science, pages 38–52. Springer-Verlag, Berlin, Heidelberg, 2011.
22. R. Young, M. Riedl, M. Branly, Jhala, A, R. Martin, and Saretto, C.J. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1), 2004.