Situating Cognitive Agents in GOLEM

Stefano Bromuri and Kostas Stathis

Department of Computer Science,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, UK
{stefano,kostas}@cs.rhul.ac.uk

Abstract. We investigate the application of a logic-based framework representing an agent environment as a composite structure that evolves over time. Such a complex structure contains the interaction between two main classes of entities: *agents* and *objects*. Interactions between these entities are specified in term of *events* whose occurrence is governed by a set of *physical laws* specifying the possible evolutions of the agent environment, including how these evolutions are perceived by agents and affect objects and processes in the agent environment. We illustrate the work using GOLEM[1], a protype platform whose aim is to implement the framework to build situated cognitive agents in a distributed agent environment.

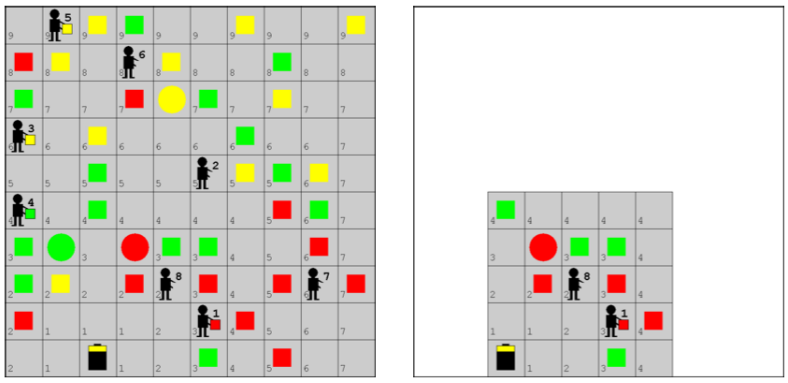
1 Introduction

It is widely acknowledged in the agent literature the need to model the agent environment in which agents are situated [1, 2, 3]. Early attempts to engineer MAS applications involved a MAS platform that implemented such an agent environment by enabling agents to interact with each other by sending and receiving messages [4, 5]. However, these early attempts in modeling the agent environment as a message transport system (or broker infrastructure) has been criticized to be inadequate for complex applications [6] requiring the treatment of an agent environment as a first class entity [7, 8].

1.1 Motivation

We are concerned with situating cognitive agents in an agent environment. Informally, by an "agent environment" we mean the virtualisation of an electronic or real environment inside an agent middleware, in such a way that agents deployed in the agent middleware can access virtual or real resources by means of standard interfaces and abstractions. As a running example, we consider the electronic environment of a virtual world called Packet-World [9]. This example has been proposed to evaluate the behaviour of Multi-Agent Systems (MAS) in which agents are explicitly situated in an environment. As shown in Fig. 1,

[1] GOLEM stands for **G**eneralised **O**nto-**L**ogical **E**nvironment for **M**ulti-agent systems



(a) A Packet-World of 10x10. (b) Local view of agent 8.

Fig. 1. The Packet-World [9].

the basic setup of the Packet-World consists of a number of differently coloured packets inside a rectangular grid, whose destination is a circle with the same colour. Each agent living in the Packet-World has a battery that discharges as the agent moves in different locations in the grid. The battery can be recharged using a battery charger. This charger emits a gradient whose value is larger if the agent is far away from the charger and smaller if the agent is closer to the charger. To locate the battery charger an agent must follow the direction of decreasing gradient values. The agents have the goal to bring the packets to the collection points and can communicate with other agents to create collaborations or to ask information about the position of the collection points.

Shortcomings of previous work In an attempt to situate cognitive agents built according to the KGP model of agency [10] we have developed in previous work the PROSOCS platform [11]. The main assumption behind PROSOCS is that an agent must have a logical mind [12] that is situated in the distributed environment of network via a body [11]. For the agent's mind PROSOCS supported a developer with the generic reasoning capabilities of KGP, which had to be programmed to allow an agent to act in the environment it was situated. For example, in the context of the Packet World, a rule of the form:

```
[ self(Picker),
  observed(see(packet(P, Colour, Position)), T),
  my_position(MyPosition),
  is_close(P, Position, MyPosition),
  destination_for(Colour, Dest)
] implies
[ assume_happens_after_once(do(Picker, pick(P)), T)].
```

would make a picker agent to perform a pick action, provided the agent has observed a packet that is close to its position and the agent knows the destination for packets of this colour. In the rest of this paper, we will refer to agents that are capable of processing this kind of rules as cognitive agents. To support this basic kind of cognition PROSOCS relied upon the CIFF proof-procedure (see [13] for details). CIFF enabled KGP agents to react and plan within the environment in which they were situated, including support for temporal reasoning. A summary of the reasoning capabilities of KGP agents and their computational characteristics, as implemented in PROSOCS, are described in [14].

PROSOCS also provided the middleware for agents to be deployed and communicate with each other by sending and receiving messages via their body. Two implementations of the middleware were developed: (a) one built on top of the JXTA peer-to-peer infrastructure and (b) another based on the TuCSON blackboard-based infrastructure. What characterised PROSOCS from other platforms of its time was that generic sensor and effector components were linked to an agent's body to enable the agent send and receive messages, including support with physical interactions between agents and objects. Experimentation with the platform [15] showed that although the development of a reusable middleware to enable communicative interaction was generally straightforward, providing general rules for the interaction between agents and objects for different applications was more a limitation than a strength. The issue here was that different applications imposed different requirements on how agents and objects need to be manipulated and coordinated. A more acceptable solution was to allow the developer to specify the low-level physical interaction for different applications, as if this developer designed the agent environment and programmed its middleware to serve the purpose of the application.

Contribution, Scope, and Significance This paper develops a logic-based framework representing an agent environment as a composite structure that evolves over time. Such a complex structure contains the interaction between two main classes of entities: agents and objects. Interactions between these entities are specified in term of events whose occurrence is governed by a set of physical laws specifying the possible evolutions of the agent environment, including how these evolutions are perceived by agents and affect objects and other agents in the environment. The emphasis of the work is to specify the representation of the agent environment declaratively, in a logic-based way, so that the programming of the agent environment is easy to understand and change. To specify what is perceived in the agent environment we use of the notion of *affordances*, to enable cognitive agents to perceive the external states of objects and other agents in order to interact with them. Through affordances a designer specifies what is possible in the agent environment at a level that can be processed directly by cognitive agents. We show how to turn the overall representation from a specification to an implementation that we call GOLEM, which is a general and reusable platform across applications and whose features are examplified by the Packet World simulation in the context of this paper. The significance

of the implemented system is that it can support complex applications through the deployment of cognitive agents situated in a distributed environment over a network.

1.2 Organisation

Section 2 introduces the general architecture of GOLEM, following the ideas presented in [16]. Section 3 shows how to represent interaction in a GOLEM agent environment on top of an extension of the *Event Calculus* based on objects [17], including any implementation issues. Section 4 places our research in the context of existing literature and compares it to related work. We summarise our effort in Section 5 where we also chart out directions for future work.

2 Description of Environment Affordances

We propose to investigate the design of the agent environment using the concept of *affordances*. This concept is normally taken to describe "all the action possibilities latent in the environment, objectively measurable, and independent of an agent's ability to recognise those possibilities" [18]. As with research in HCI [19], we rely upon *perceived affordances* where entities of an environment "suggest" to agents (whether artificial or human) how they should interact with them. In other words, we do not expect our agents to learn how to interact with an object by randomly taking actions [20] according to previous experience[21]. Instead, we propose an agent's environment to be designed in advance, assuming a particular ontology, very much like an interactive system, with the aim to treat cognitive agents like we treat users. This does not prevent an agent from learning how to use the object, because knowing the interface of the object, the agent could just try to explore the functionality by observing an action's effect on the agent environment.

We answer what the developer needs to design by relying on the conceptual framework described in [16]. This defines an agent environment as a *container* where *agents* interact with other agents and *objects* using *sensors* and *effectors*. We expand this preliminary work by providing a framework stating how to specify logically these entities and their interaction using *events*. Events describe what happens in the agent environment as a result of actions being executed by effectors. According to the happening of an event the agent environment notifies those sensors capable of perceiving the action of the event. For the purposes of this paper we distinguish between three types of acts embedded in an event: *speech acts* - to allow agents to communicate with other agents and users; *sensing acts* - to allow an agent to perceive the environment actively; and *physical acts* - to allow the agent to interact with other entities, in particular objects, but also agents as well. To simulate these acts we will rely upon different kinds of sensors and effectors the agent should possess to capture the interaction in the agent environment. Our primary concern is to provide a computable specification of the interaction rather than a formal definition; the latter is beyond the scope of this paper.

2.1 Objects

GOLEM uses a particular architecture for objects shown in Fig. 2. As part of this architecture the object is described in terms of the perceived affordances. To present these perceived affordances we use the object-based notation used by C-logic [22], a formalism that allows the description of complex objects. A description of the form:

```
packet: p1[ colour ⇒ red,
            methods ⇒ {pick, drop, hit},
            position ⇒ square:sq1,
            receptors ⇒ { receptor:r1 },
            emitters ⇒ { emitter:em1 }
          ]
```

states that p1 is a complex term of class packet, with a functional attribute describing that the colour is red, a multi-valued attribute methods stating that the actions afforded by the object the term represents are pick, drop, and hit, a functional attribute asserting that the position of the packet is in square sq1, a multi-valued attribute receptors containing one receptor sensor r1, and a multi-valued attribute emitters containing one emmiter effector em1. Some of the attribute values are complex terms themselves, for example, sq1 is a complex term containing information such as the coordinates of the packet in the Packet-World grid. The C-logic syntax to represent the perceived affordances of an object as a complex term has a first-order logic translation, as we can see for a packet object below:

is_a(p1, packet).	method_of(p1, hit).	attribute(packet, colour, single).
colour(p1, red).	position(p1, sq1).	attribute(packet, method, multi).
method_of(p1, pick).	receptor_of(p1, r1).	attribute(packet, receptors, multi).
method_of(p1, drop).	emitter_of(p1, em1).	attribute(packet, emitters, multi).

In this way, we represent all the related information that is perceived of an object, including its relationship with other entities in the agent environment.

The idea behind having receptor sensors for an object is that they receive notifications from the agent environment as a results of actions executed on that object. In general, the receptor sensor of an object can only capture notifications of *physical acts* performed on the object by entities in the agent environment that are capable of executing these actions. To represent events that receptors can capture we use complex terms too. The term:

```
do:e1 [actor ⇒ agent:ag1 [effector ⇒ ef1], act ⇒ hit, object ⇒ packet:p1]
```

describes an event e1 where the effector ef1 of agent ag1 performs a physical act hit on packet p1. Such an event will be captured by the receptor of the object via notification sent to the object by the environment. Then the object's *processor* will call a *method* of the *internal object*. The general idea behind the internal object is that it wraps in it a resource of the external environment, thus

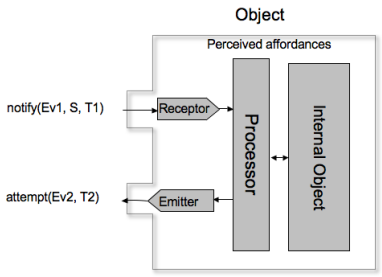


Fig. 2. A GOLEM object whose receptor S is receiving a notification of an event Ev1 at time T1 and whose emitter attempts to make event Ev2 happen at time T2.

hiding from the agents the complexity of interfacing with the external resource. In other words the object abstraction can be a virtual entity, as for objects in Packet World, or a virtualisation of an external resource of the external real environment. The method call will typically result in the output of the call transmitted as another event via the object's *emitter* effector. As before, emitted events are complex terms. To simulate a packet's reaction to the physical act represented by e1, the event description:

hearing:e2 [emitter ⇒ packet:p1 [effector ⇒ em1], sound ⇒ packet_hit]

showing the kind of event emitted by the object. Events may be emitted by the processor also upon conditions determined entirely upon the state of the internal object and not necessarily as a reaction to an external trigger. The details we omit as these events can be described similarly, the only part that changes is the type and content of the event.

2.2 Cognitive Agents

GOLEM agents are organised as an extension of the PROSOCS anthropomorphic architecture of an agent [11], shown in Fig. 3. In this architecture an agent has a *body* whose affordances can be perceived by other agents. A description of the form:

```
picker: ag1[ understands ⇒ ontology:o1,
             sensors ⇒ {sight:s1, hearing:s2, smell:s3},
             effectors ⇒ {speak:ef1, arm:ef2, arm:ef3},
             position ⇒ square:sq3,
             activity ⇒ idle
           ]
```

states that ag1 is a packet picker understands the ontology o1 (of packet world), has sensors of class sight, hearing and smell, and effectors of class speak and arm,

its position is square sq3 in the container, and it is currently idle. The position of the agent describes a set of relative terms relating the agent with other entities in the agent environment. As with objects, the effectors of an agent attempt to execute physical actions in the agent environment. Similarly, agent sensors respond to event notifications by the agent environment. These notifications enable an agent's sensors to *passively* observe the agent environment [10]. Alternatively, sensors actively observe the agent environment through sensing acts, giving rise to active observations [10]. Active observation is expressed as a sensing act that attempts to perceive certain properties of the agent environment. For example, the term below shows how agent ag1 focuses on a specific part of the agent environment:

sensing:e3 [actor⇒ ag1[sensor⇒ s1], act ⇒ look, focus ⇒ p1[color⇒ X]]

by looking with sensor s1 to find the colour of packet p1, denoted by the variable X. The outcome of such a request will result in an asynchronous call to the agent environment to return the variable substitution, as we will see later in section 3.2.

Apart from situating the agent in the agent environment, the body contains a *brain* to connect the various sensors attached to it. The brain also provides an interface to the *mind*, a cognitive component giving the agent the ability to reason logically and make decisions. This mind-brain separation allows different cognitive models of agency to be interfaced to the body, thus making the architecture more flexible. From an agent environment perspective, a user can use an agent's body to access the electronic environment, in which case the brain of the agent provides simply a convenient interface for the user to select actions using his own mind.

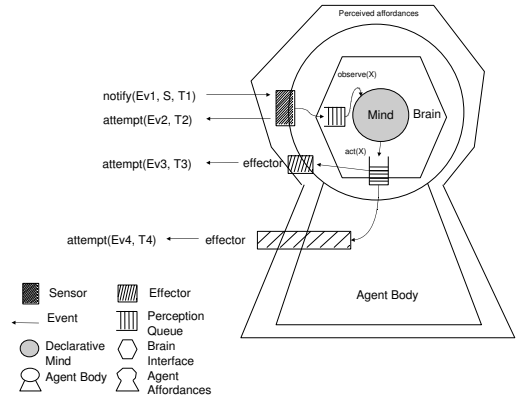


Fig. 3. The anthropomorphic agent architecture in GOLEM (adapted from [11]).

2.3 Containers

An agent environment in GOLEM is a first class entity referred to as a set of *containers*. As shown in Figure 4 the container has a *state* that acts as a directory of all the present agents and objects in it, including information about their topology and configuration. Interactions between the entities of an agent environment are governed by a set of *physical laws*. These laws specify the possible evolutions of the container, including how these evolutions are perceived by agents and affect objects and processes in the environment.

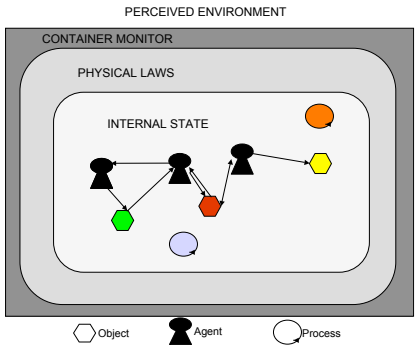


Fig. 4. A GOLEM Container

As with agents and objects, the container has its own perceived affordances that include the ways in which an agent can configure itself (or other basic, object, agent, and containers) to became part of the container's internal state. For example the description:

```
container:c1[address ⇒ "container://one@134.219.7.1:13000",
             laws ⇒ physics:pw1,
             type ⇒ open,
             entities ⇒ {agent:ag1, packet:p1, packet:p2, destination:d1, battery:b1}
            ]
```

describes a container whose address is container://one@134.219.7.1:13000, its laws are represented by another object pw1 of class physics, it is an open container in that any agent can enter it, and whose internal state contains five entities, one agent (ag1), two packets (p1, p2), a destination for packets (d1), and a battery (b1). Before an agent enters the container it can inspect the laws attribute containing the physics for the Packet-World, further specified as:

```
physics:pw1[name ⇒ "PacketWorld"
```

```
 mediates ⇒ {see, speak, listen, do},
 entities ⇒ {agent, object},
 processes ⇒ {pheromon_evaporation},
 ontology ⇒ {"PacketWorldOntology"}
]
```

By examining a physics term such as pw1 above, an agent can perceive the container by looking at the classes of events a container mediates, and other information regarding the kind of entities that the container contains, and the ontology specifying the features of these entities.

3 Interactions in a GOLEM Environment

The main task of our work has been to describe an agent environment in a form that is usable by the cognitive agents situated in it. So far we have discussed how a domain application can be described in terms of the perceived objects and agent bodies that are part of a container that acts as the agent environment. We have also shown how such a container can be represented as a complex term. In this section, we show how to describe the evolution of a container as an event calculus theory extended with a part that enables objects and agents to interact. We close the discussion with a summary of our implementation.

3.1 The agent environment and its evolution

To represent how phenomena change the state of a GOLEM container we use the object-based event calculus (OEC) described by Kesim and Sergot in [17]. The OEC extends the data model of the original event calculus with one that describes how instances of complex terms evolve over time. This framework allows the developer of a GOLEM application to specify the effects of actions to/from objects and agents as events. A subset of the clauses describing the OEC is given in Fig. 5.

Clauses C1-C2 provide the basic formulation of OEC deriving how the value of an attribute for a complex term holds at a specific time. Clause C3 describes how to represent derived attributes of objects treated as method calls computed by means of a solve_at/2 meta-interpreter as specified in [23]. C4-C5 support a monotonic inheritance of attributes names for a class limited to the subset relation. As C1-C2 describe what holds at a specific time, C6-C7 determine how to derive the instance of a class at a specific time. The effects of an event on a class is given by assignment assertions; the clause C8 states how any new instance of a class becomes a new instance of the super-classes. Finally, deletion of objects is catered for by clauses C9-C11. C9 deletes single valued attributes that have been updated, while C10-C11 delete objects and dangling references.

To describe how the affordances in the agent environment evolves as a result of events happening in it we need to define domain specific initiates and terminates clauses. For example, to describe an agent moving in the Packet-World grid, we write:

(C1) holds_at(Id, Class, Attr, Val, T)←
happens(E, Ti), Ti ≤ T,
initiates(E, Id, Class, Attr, Val),
not broken(Id, Class, Attr, Val, Ti, T).
(C2) broken(Id, Class, Attr, Val, Ti, Tn)←
happens(E, Tj), Ti < Tj ≤Tn,
terminates(E, Id, Class, Attr, Val).

(C3) holds_at(Id, Class, Attr, Val, T)←
method(Class, Id, Attr, Val, Body),
solve_at(Body, T).

(C4) attribute_of(Class, X, Type)←
attribute(Class, X, Type).
(C5) attribute_of(Sub, X, Type)←
is_a(Sub, Class),
attribute_of(Class, X, Type).

(C6) instance_of(Id, Class, T)←
happens(E, Ti), Ti ≤ T,
assigns(E, Id, Class),
not removed(Id, Class, Ti, T).
(C7) removed(Id, Class, Ti, Tn)←
happens(E, Tj), Ti < Tj ≤ Tn,
destroys(E, Id).

(C8) assigns(E, Id, Class)←
is_a(Sub, Class),
assigns(E, Id, Sub).

(C9) terminates(E, Id, Class, Attr, _)←
attribute_of(Class, Attr, single),
initiates(E, Id, Class, Attr, _).

(C10) terminates(E, Id, _, Attr, _)←
destroys(E, Id).
(C11) terminates(E, Id, _, Attr, IdVal)←
destroys(E, IdVal).

Fig. 5. A subset of the *Object-based Event Calculus* from [17]

initiates(E, picker, A, position, Pos, T) ←
do:E [actor ⇒ A, act ⇒ move:M [destination⇒ Pos]].

To complete with describing the effects of the event we also need to terminate the attribute holding the old position of the agent, in this case, this is handled by the general rule described in clause C9.

3.2 Representation of Interaction

Given the OEC to support the evolution of the agent environment's state we use on top of it a set of logic programs that work together with the event calculus, to represent the interactions in a GOLEM environment. In what follows, we are presenting extracts of our formulation, to exemplify the approach.

Action Execution As we discussed in section 2, the execution of actions in GOLEM are represented as attempts. Attempts are the same as what Ferber [24] calls influences, we prefer the use of attempt because it captures better our intention, namely the action that is about to occur as an event in the agent environment. Attempts are described by assertions of events at a specific time. We keep the description of events separately from attempts. Suppose for instance that an agent (ag1) is attempting to make a move to square sq3 at time 120. In GOLEM this will be represented by an attempt as shown below:

attempt(e14, 120).
do:e14 [actor ⇒ ag1, act ⇒ move:m1 [destination⇒ sq3]].

Such an attempt causes the event of moving to happen, provided the event described in the attempt is possible according to the physics of the agent environment. There are two ways we propose to define this:

```
(H1) happens(Event, T)←
         attempt(Event, T),
         possible(Event, T).
```

```
(H2) happens(Event, T)←
         attempt(Event, T),
         not impossible(Event, T).
```

Definition H1 suggests that we must describe for every agent environment when an event is possible at a specific time. Often, as the number of events that happen is large, H2 suggests that it would be easier if we described what events are impossible at a specific time. Depending on the application, the developer of an agent environment can choose between H1 or H2. In the Packet-World, for example, we have found easier to describe what is impossible rather than what is possible, and rely upon the use of negation-as-failure to handle what is possible by default. As an example of an impossible event description, consider how to define what is impossible when an agent attempts to move to a square in the grid that is occupied already:

```
impossible(E, T)←
      do:E [actor ⇒ A, act ⇒ move:M [destination⇒ Pos]],
      holds_at(Pos, square, status, occupied, T).
```

We need to define similarly additional impossible/2 constraints of this kind to deal with situations where an agent is trying to move outside the grid, for example. Impossibility constraints can also be used to handle more than one event attempted at the same time, thus making the approach quite expressive.

Using the definition H1, a developer has also the option to combine possible/impossible constraints if the following general rule is added:

```
possible(E,T)← not impossible(E,T).
```

This new definition makes H1 more general, since the developer is now in a position to specify both domain specific rules of both what is possible or what is impossible, case by case, thus allowing representations that are more expressive.

Passive Perceptions When an event happens, it is notified instantaneously to all types of sensors that are capable of detecting it. Put another way, certain types of sensors will be filtering out specific kind of perceptions. This fact is reflected in the definition of event notification that takes into consideration the type of event that happens. For passive perceptions we need to check that the event does not contain a sensing action, so the notification is defined as:

```
notify(E, S, T) ←
     happens(E, T),
     not sensing(E),
     detectable(E,S,T)
     not interfered(E, S, T).
```

We assume that event types contain, as part of their description, the sensor types that can detect it. We use the notion of *detectable* as possibility for percepts. For the packet world we define it as:

```
detectable(E,S,T) ←
        E [is_detected_by ⇒ SensorClass],
        instance_of(S, SensorClass, T),
        holds_at(S, SensorClass, status, open, T).
```

The definition of notify/3 also checks that when an event is notified it is not interfered by an obstacle. Interference is a domain specific constraint that for some applications may remain undefined. To exemplify it in the Packet-World, we try simulate the fact that some events will not be possible to perceive because there is an entitiy (object or agent) that hides its occurrence. To do this we define the following rule:

```
interfered(E, S, T) ←
        E [coordinates ⇒ XYe],
        instance_of(S, sight, T),
        holds_at(A, picker, sensor_of, S, T),
        holds_at(A, picker, coordinates, XYa, T),
        holds_at(Entity, entity, coordinates, XYent, T),
        in_between(XYent, XYa, XYe).
```

In other words, a notification is interfered only when there is an entity between the position of the agent and the location in which the event happened.

Active Perceptions Agents in GOLEM are enabled to actively perceive objects in the agent environment. Such perceptions assume that the agent has attempted to perform a sensing act with a specific focus query for the object. This is initiated by an attempt of an sensing act with a particular focus. We specify this as:

```
perceive(E, S, T) ←
        happens(E, T),
        sensing(E),
        detectable(E, S, T),
        E [sensor_of⇒ S, focus⇒ Focus],
        solve_at(Focus, T).
```

The call to solve_at is assumed to be an asynchronous call to the agent environment which returns the variable substitutions to the Focus, if any. It is important to note that the time T is not instantiated by the agent who is trying to perceive, but by the agent environment who receives the call.

3.3 Implementation issues

We have implemented GOLEM according to the reference model of Fig. 6. In this figure actions coming from containers, agents, objects, or internal *Processes*, are collected by an *Attempts* module. Attempts of action are mediated by a *Physics* component ensuring that these actions are possible before they happen as events in the state. The physics module is in charge to mediate the three kind of events described in section 2. As a consequence, physics acts, speech acts and sensing acts are mediated before taking place in the agent environment, or, in other words, the agent environment allows to define laws of interaction for these three kind of events. The physics also describe how events cause changes to the perceivable state of the agent environment. Once an event has happened, it is directed by the notification module to the *Passive Perception* module that notifies the sensors of agents and objects.

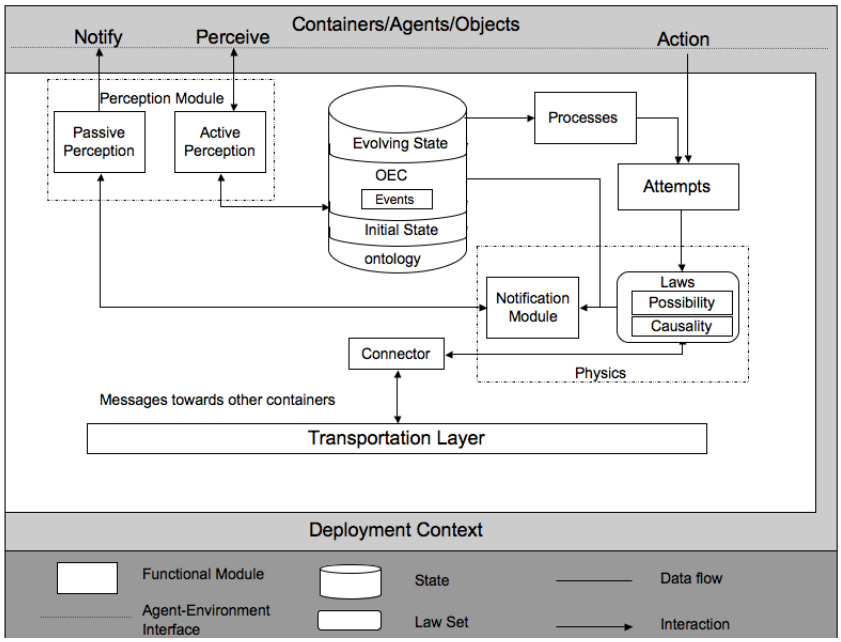


Fig. 6. The GOLEM Reference Model.

Active perceptions of agents on objects are handled by the *Active Perception* module that accesses the state of the agent environment to support the requested perceptions. Containers are recursively deployed as objects, so that the agents in the agent environment can access a container from another container. The use of a *Connector* component allows the agent environment to forward/receive messages to/from other containers via the transportation layer.

We have implemented our framework according to the above reference model using tuProlog [25] and Java. Using this combination we use Java to implement Agents, Objects and the Container. The container has inside a Physics component that uses tuProlog to define the logic-based agent environment. To implement the specification we need to slightly change some of the rules specified earlier. For example, the rule H1 is rewritten so that attempts become agent environment calls that assert event descriptions in the state of the agent environment:

```
attempt(Event, T):- not impossible(Event, T), add(happens(Event, T)).
```

Agents, objects, containers, or internal processes will instantiate the Event at the time of the call, while the time T is instantiated by the agent environment. add/2 asserts separately the happening of the event from the event's description.

Other features of the tuProlog/Java combination include allowing a developer to support asynchronous communication and primitives to register Java objects inside a Prolog context, using the Java Reflection API [26]. We use these facilities to define declaratively how to deploy agents, objects, agent sensors and services to create the GOLEM distributed environment. Defining the rules of the agent environment using a Prolog theory is particularly helpful when a developer needs to change the interaction inside a container. With the GOLEM's toolset, we allow a platform administrator to open a container, inspect it, and subsequently change the physical laws governing it. There are a number of issues that we have to take into consideration here, in particular, ensuring consistency of the physics and the atomicity of action execution. A detailed discussion of these issues, however, is beyond the scope of this paper.

To allow a container's affordances to be discovered within a distributed environment, we translate our complex terms describing a GOLEM container to WSMO [27] ontologies and concepts. This mapping is straightforward as there is a syntactic link between OEC and F-logic[28] upon which WSMO relies. For example, a picker agent description in GOLEM can be translated to a WSMO concept as follows:

```
concept Picker subConceptOf Agent
    UnderstandOntology ofType (1 1)_iri
    hasSensors ofType Sensor
    useEffectors ofType Effector
    hasPosition ofType Square
    hasActivity ofType (1 1)_string
```

The motivation behind the use of WSMO is to use it as a standard for allowing agents from other platforms to discover and use resources of GOLEM. A cognitive agent that looks at the affordances of an entity, knows immediately the messages to interact with the entity, as well as its observable properties. Fig. 7 shows an example of execution where GOLEM entities and their affordances are described in WSMO.

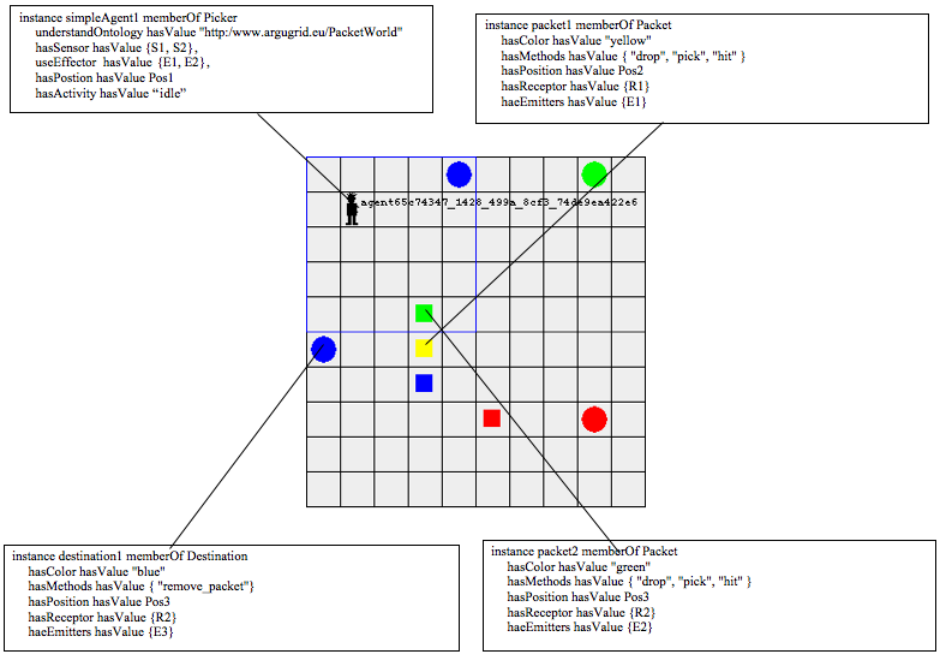


Fig. 7. A container node with Packet World inside

4 Related Work

There is a growing research and development effort on how to model situated multi-agent systems, see [6] for a discussion. Our work is inspired by the *influence-reaction* model by Ferber and Müller [2] and its extensions as formalised by the work of Weyns and Holvoet [29]. In our framework influences are represented as attempts of events and reactions as environment notifications. However, in this work we are not concerned with synchronisation issues as [2] and [29] but rather with how to specify interaction in computational logic, thus providing executable specifications of agent environments. Despite an apparent similarity of our container with the description of the agent environment in [8], at a closer look the two approaches rely upon different reference models.

Our representation of active perception relates naturally to the work of Weyns et al [30] who divide an agent's perception in three parts: *sensing, interpreting* and *filtering*. Our work is really about the first part, namely, the mapping of the external environment to a symbolic representation suitable for the agent using what we called *sensing* acts. As in Weyns et al, the agent can select a set of *foci* that enables an agent to direct its perception and perceive only specific types of information, simulating a kind of artificial sight for agents. The interpretation part of the perception mechanism of Weyns et al maps the representation of the

agent environment in the actual percept of the agent. These percepts have the function to describe the sensed agent environment in the language understood by the agent. In our approach, we have tried to minimize interpretation and standardize it to be logical terms. We have left filtering outside the framework as this part concerns the way sensors work, which is beyond the scope of this work.

Vizzari in [31] models the concept of environment as a multi-layer multi-agent situated system (MMASS). The environment is composed by a set of graphs interconnected by interfaces, forming thus a multilayered structure with some interfaces among layers. Every layer, and thus every graph, may represent a specific aspect of agents' environment: for instance one of them may represent an abstraction of agents' physical environment, while other ones may be related to other conceptual topologies such as organization charts or dependency graphs. In GOLEM, instead of defining layers, we define rules. Different sets of rules can then describe different layers of the agent environment. What we have presented here is only a framework for the physical interaction where attempts for action result in events, which for Vizzari generate fields, signals capable to diffuse through the layers, according to the interfaces between these layers. In addition, signals in Vizzari's framework can be perceived by agents according to specific rules of perception based on functions such as *diffusion, composition* and *comparison*. For us diffusion is notification, composition is complex term creation, while comparison is our use of having different sensors capturing different types of events.

The coordination artifact theory [32] defines as an abstract model that takes inspiration from concrete objects supporting the interaction of physical entities. Agents perform their activities in the environment helped by coordination artifacts, generally passive entities that defines a usage interface, a set of operating instruction and a coordination behaviour specification. For an agent to understand how to interact with an artifact one has to understand the interfaces of that artifact. GOLEM follows the TuCSoN idea that the infrastructure must be programmable. While coordination artifacts for agent interaction take inspiration from actual concrete objects of the real world, our approach brings the metaphor of agent environment to the extreme by taking into account spatio-temporal features. These features are represented (possibly in an explicit way) and have an influence on perception, interaction and as a result on agent behaviour.

A more recent technology supporting the coordination artifact model is called CartAgO, proposed by Ricci et al in [33], which is in the process of being integrated with Jason [34]. This technology proposes a model of perception and actions that is similar to the one proposed by Weyns et al in [30]: Agents can have sensors that perceive well-defined kind of perceptions and filter them at runtime. The notion of *workspaces* contain artifacts and agents, used also to define the topology of the working environment. Through workspaces it is possible to model a notion of locality, in terms of the artifacts that an agent can use and observe.

There are many similarities of GOLEM with CArtAgO: they both use sensors and effectors for agents, as PROSOCS did [11], CArtAgO workspaces correspond to GOLEM containers, and as in CArtAgO we distinguish between speech acts, physical acts, and sensing acts. Moreover the Jason integration offers the possibility to defines user defined agent environments specifying pre-conditions, post-conditions and effects of the action in the environment, as well as offering a language to define BDI agents acting in the agent environment. However, there are many differences as well, the most important being that in GOLEM we keep the rules of the physical environment in the container, not in the artifacts, and we expect the implementation to enforce them in a distributed manner. Finally, instead of manuals keeping operating instructions for artifacts GOLEM uses affordances.

Affordances are also strongly related with the work reported by Platon et al. in [35], [36], and [37]. As in PROSOCS, this work puts forward the use of an *agent soft body* which has a state that is public and available to an observer. The act of observing such a state in the Platon et al. framework is based on the notion of *oversensing* [35] and *overhearing* [36]. In our work the oversensing/overhearing acts are modelled as active perception on the affordances of environment entities. Other differences with the Platon et al. work are that GOLEM affordances express more than a simple state, they express also the interaction interface of both agents and objects, rather than only agents.

5 Conclusions

We have presented a logic-based framework representing an agent environment as a composite structure that evolves over time. Such a complex structure contains *agents* and *objects* in *containers*, whose interaction is specified in term of *events*. Occurrence of events is governed by a set of *physical laws* specifying the possible evolutions of the environment, including how these evolutions are perceived by agents and affect objects and processes in the environment.

We have implemented the framework in GOLEM, a prototype platform exemplified here using the Packet-World. The benefits of our approach can be summarised as follows. By using a declarative approach we define the rules that constrain the interactions in an agent environment and then update them at run time, without the need to restart the application (an important issue if we want to incrementally introduce patches to an application environment). We do not need to translate the perceptions from the environment to the mind of the agent as the agent environment and mind of an agent use the same representation language, thus making the situating of cognitive agents easier. By introducing the idea of affordances and wrapping external resources in objects we hide the complexity of how an agent can interact with the external world; knowing the affordances of an object the agent has the interface of that object standardised by the use of ontologies. Finally, by keeping a history of events we can easily playback interactions and therefore debug an application through a log, in the case

that the agent environment models a simulation that does not involve external resources, wrapped in the object abstraction.

We are currently studying the benefits of our approach in the ArguGRID project [38], where the mind of the agent is defined using argumentation [39]. Now interaction with objects is interaction of agents and/or users with semantic web-services defined in WSMO. As part of this work we are seeking to build upon the lemma generation mechanism discussed in [23] to improve the scalability of the GOLEM's approach.

Acknowledgments

This work was partially supported by the IST-FP6-035200 ArguGRID project. We would like to thank Danny Weyns and the anonymous revievers for their comments in a previous version of this paper.

References

[1] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edn. Pearson Education (2002)
[2] Ferber, J.: Multi-Agent Systems. Addison-Wesley, Harlow (1999)
[3] Kowalski, R.A.: Reconciling Logic and Objects. In: 6th Mexican Intl. Conference on Computer Science (ENC), IEEE Computer Society (2005)
[4] JADE: Java Agent DEvelopment framework Home Page: http://jade.tilab.com.
[5] Nwana, H.S., Ndumu, D.T., Lee, L.C., Collis, J.C.: ZEUS: A Toolkit for Building Distributed Multiagent Systems. Applied Artificial Intelligence **13**(1-2) (1999) 129–185
[6] Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for Multiagent Systems State-of-the-Art and Research Challenges. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: E4MAS. Volume 3374 of Lecture Notes in Computer Science., Springer (2004) 1–47
[7] Odell, J., Parunak, H.V.D., Fleischer, M., Brueckner, S.: Modeling Agents and their Environment: The Physical Environment. Journal of Object Technology **2**(2) (2003) 43–51
[8] Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems **14**(1) (2007) 5–30
[9] Weyns, D., Helleboogh, A., Holvoet, T.: The Packet-World: a Test Bed for Investigating Situated Multi-Agent Systems. Software Agent-Based Applications, Platforms and Development Kits, Whitestein Series in Software Agent Technology. (2005)
[10] Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of Agency. In: Proceedings of the 16th European Conference of Artificial Intelligence, Valencia (2004) 33–37
[11] Stathis, K., Kakas, A.C., Lu, W., Demetriou, N., Endriss, U., Bracciali, A.: PROSOCS: a platform for programming software agents in computational logic. In Müller, J., Petta, P., eds.: Proceedings of the 4th Intl. Symposium "From Agent Theory to Agent Implementation" (AT2AI-4), Vienna (April 13-16 2004) 523–528

[12] Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Stathis, K.: Crafting the Mind of a PROSOCS Agent. Applied Artificial Intelligence **20**(4-5) (2006) 105–131
[13] Endriss, U., Mancarella, P., Sadri, F., Terreni, G., Toni, F.: Abductive Logic Programming with CIFF: System Description. In Alferes, J.J., Leite, J.A., eds.: Proceedings of Logics in Artificial Intelligence, JELIA 2004, Springer (2004) 680–684
[14] Bracciali, A., Demetriou, N., Endriss, U., Kakas, A.C., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP Model of Agency for Global Computing: Computational Model and Prototype Implementation. In Priami, C., Quaglia, P., eds.: Global Computing, (GC 2004). Volume 3267 of Lecture Notes in Computer Science., Springer (2004) 340–367
[15] Stathis, K., Toni, F.: Ambient Intelligence using KGP Agents. In: European Symposium on Ambient Intelligence (EUSAI04). Volume 3295 of LNCS., Springer (September 2004) 351–362
[16] Stathis, K., Kafetzoglou, S., Papavasilliou, S., Bromuri, S.: Sensor Network Grids: Agent Environments combined with QoS in Wireless SensorNetworks. In: The 3rd Intl. Conference on Autonomic and Autonomous Systems (ICAS07). (Jun 2007)
[17] Kesim, F.N., Sergot, M.: A Logic Programming Framework for Modeling Temporal Objects. IEEE Transactions on Knowledge and Data Engineering **8**(5) (1996) 724–741
[18] Gibson, J.J.: The Ecological Approach to Visual Perception. Lawrence Erlbaum Associates (1979)
[19] Norman, D.A.: Affordance, Conventions, and Design. Interactions **6**(3) (1999) 38–43
[20] Child, C., Stathis, K.: Rule Value Reinforcement Learning for Cognitive Agents. In Nakashima, H., Wellman, M.P., Weiss, G., Stone, P., eds.: 5th Intl. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), ACM (2006) 792–794
[21] Sequeira, P., Vala, M., Paiva, A.: What can I do with this? Finding possible interactions between characters and objects. In: Proceedings of the 6th Intl. Conference of Autonomous Agents and Multi-agent Systems (AAMAS 2007), IEEE Computer Society (2007)
[22] Chen, W., Warren, D.S.: C-logic of Complex Objects. In: PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, New York, NY, USA, ACM Press (1989) 369–378
[23] Kesim, N.: Temporal Objects in Deductive Databases. PhD thesis, Imperial College (1993)
[24] Ferber, J., Müller, J.P.: Influences and Reactions: a Model of Situated Multiagent Systems. In: ICMAS'96 (Intl. Conference on Multi-Agent Systems), AAAI Press (1996)
[25] tuProlog: tuProlog Home Page: http://www.alice.unibo.it:8080/tuProlog/.
[26] Denti, E., Omicini, A., Ricci, A.: Multi-paradigm Java-Prolog integration in tuProlog. Science of Computer Programming **57**(2) (August 2005) 217–250
[27] WSMO: Web Service Modelling Ontology Home Page: http://www.wsmo.org/.
[28] Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the Association for Computing Machinery (May 1995)
[29] Weyns, D., Holvoet, T.: A Formal Model for Situated Multi-agent Systems. Fundam. Inform. **63**(2-3) (2004) 125–158

[30] Weyns, D., Steegmans, E., Holvoet, T.: Towards Active Perception in Situated Multi-Agent Systems. Applied Artificial Intelligence **18**(9-10) (2004) 867–883
[31] Vizzari, G.: Dynamic Interaction Spaces and Situated Multiagent Systems: from a Multilayered Model to a Distributed Architecture. PhD thesis, University of the Studies of Milan Bicocca (2003-2004)
[32] Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination Artifacts: Environment-based Coordination for Intelligent Agents. In: Autonomous Agents and Multi-agent Systems, Washington, DC, USA (2004) 286–293
[33] Ricci, A., Viroli, M., Omicini, A.: CArtAgO: A Framework for Prototyping Artifact-based Environments in MAS. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: Environments for MultiAgent Systems III. Volume 4389 of LNAI. Springer (February 2007) 67–86
[34] Hübner, J.F., Bordini, R.H.: Jason, a java-based interpreter for an extended version of agentlink. http://jason.sourceforge.net/
[35] Platon, E., Sabouret, N., Honiden, S.: Oversensing with a softbody in the environment - another dimension of observation. In: Proceedings of Modelling Others from Observation'05. (2005)
[36] Weyns, D., Parunak, H.V.D., Michel, F., eds.: Environments for Multi-Agent Systems II, Second International Workshop, E4MAS 2005, Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Papers. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: E4MAS. Volume 3830 of Lecture Notes in Computer Science., Springer (2006)
[37] Platon, E., Sabouret, N., Honiden, S.: Tag interactions in multiagent systems: Environment support. In Gleizes, M.P., Kaminka, G.A., Nowé, A., Ossowski, S., Tuyls, K., Verbeeck, K., eds.: EUMAS, Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten (2005) 270–281
[38] ArguGRID: ARGUmentantion as a foundation for the semantic GRID. http://www.argugrid.eu/
[39] Morge, M., McGinnis, J., Bromuri, S., Toni, F., Mancarella, P., Stathis, K.: Towards a Modular Architecture of Argumentative Agents to Compose Services. In: Proc. of the of 15th Journees Francophones sur les Systemes Multi-Agents (JFSMA), Carcasonne, France (Nov 2006)