A Virtual E-retailing Environment in GOLEM

Stefano Bromuri, Visara Urovi, Pedro Contreras, Kostas Stathis

Department of Computer Science. Royal Holloway, University of London
{stefano, visara, pedro, kostas}@cs.rhul.ac.uk

Abstract

We present a prototype multi-agent system whose goal is to support a 3D application for e-retailing. The prototype demonstrates how the use of agent environments can be amongst the most promising and flexible approaches to engineer e-retailing applications. We illustrate this point by showing how the agent environment GOLEM uses semantic web concepts to develop the e-retailing application. In this context we describe the features of GOLEM that allow a user to become an avatar and explore the environment by searching and dynamically discovering new products and services.

Keywords: Semantic Web Services, 3D Agent Environment, GOLEM.

1 Introduction

E-retailing is a common and widespread class of applications used by retailers to sell products and services to users on the Web. Although such class of applications is developed with traditional web-based tools and techniques, there is increasing interest to revisit them by using (a) Semantic Web technologies [2] to allow computer programs support for (semi) automatic selling for retailers or buying for users, and (b) virtual environments such as Second Life [11] to provide users with a more engaging experience to the existing home page metaphor supported by HTML browsers.

We present a new way of looking at the interaction for e-retailing that is engineered using the notion of agent environment [15] as a first-class abstraction. We extend the agent environment abstraction to support the deployment and discovery of businesses and generally e-retailing services visualised by a 3D virtual environment interface. Our developed prototype uses GOLEM (**G**eneralized **O**nto-**L**ogical **E**nvironment for **M**ulti-agent systems) [4] as the agent environment. GOLEM supports the semantic description of services and interactions among user, agents and services.

The significance of the implemented prototype is that it illustrates how agent technology based on 3D virtual agent environments can be used to develop the next generation e-retailing. The work contributes showing how by using GOLEM, e-retailing services are deployed as shops of a 3D scene and semantically localised through their descriptions. Support to e-retailing is then a consequence of the interaction between agents in charge of services and users represented via agent avatars. With respect to popular technlogies like Second Life, which hard code the interaction using a proprietary language that has both the responsibility to deal with the graphical part and with the interaction, our prototype separates the visualisation part from the logical part of the virtual environment, embedding the graphical part in a standard ontological description based on the WSML language [18], and dealing with the interaction according to the agent environment metaphor. This brings the advantage that if the visualisation part changes, the underlying system can remain the same.

The remainder of this paper is organised as follows. In section 2 we present the use case which motivates our prototype. In section 3 we discuss a GOLEM extension to deal with e-retailing services, agents, objects and the 3D visualisation. In section 4 we describe how the interaction takes place inside the agent environment. In section 5 we show a virtual environment execution relying on GOLEM. Section 6 presents related work and section 7 presents conclusions and future work.

2 Motivation: E-Retailing Use Case

Our e-retailing application is motivated by a use case scenario exemplified by the situation where a user wants to buy a mobile phone. In this context, usually the user visits the mobile phone web-site and looks for the nearest mobile shop. There are four problems to overcome with this setting.

The discovery problem: the user needs to locate in a map the nearest shops that sell a particular brand of mobile phones. The current practice is to use a search engine which relies on a keyword based algorithm. As keyword based search is not intended to be semantic, the search engine would return a large item list with the word "mobile" in it. The overall result is that the user spends a lot of time to understand which shops sell what he really needs.

The best route problem: provided that the required shop is found the user may want to find the best route to visit it. Assuming that the user may not have a car or may have mobility difficulties the cost of the route may be considerably different according to the mobility requirements of the user. We assume a virtual environment that faithfully reflects a real one. We consider a database organised in such a way that roads and shops are semantically annotated with costs related to the mobility of the user, thus the discovery algorithm is more general and more extendible. Moreover the route provided by the search engine does not provide reference points which the user may use when trying to reach a destination. Despite the route is provided, due to this lack of reference points, the user can get completely lost when try-

ing to reach the destination in the real world. In this case, a 3D virtual environment would enhance the user experience, allowing him to have a better understanding of the route.

The requirement problem: once the shops are discovered and reached in the virtual environment, the user may not know exactly which product to buy. In this case a virtual environment with shopkeeper agents may help with information or selling products. In particular, if the user is a frequent buyer in the virtual shop, the shopkeeper may remember him and propose some good deals according to the previous interaction history.

The on demand personalisation problem: the actual browser by definition has no intelligence embedded in it. As a result the whole interaction is lead by the user searching for resources without any real feedback. In our view the experience of the user may benefit from a browser profiling him also when he is off-line, representing him as an active and persistent avatar in the virtual environment.

3 3D E-retailing in GOLEM

GOLEM is a framework to design agent environments enabling users or software agents to interact over a distributed network. The resulting multiagent system environment can be annotated semantically so that agents can perceive the interactions and act upon them to satisfy their goals. The semantic annotations model the notion of affordances as presented in [6], to capture the idea that objects and agents are perceivable in the agent environment, and that their semantic description "suggests" how to interact with them and also presenting their current state.

3.1 Container Logic Architecture

In GOLEM a container represents an atomic part of the agent environment, which is distributed as a GRID of containers. Containers provide the support for deploying agents, objects, processes and environmental services inside a declarative context, which constraints the interaction of the entities according to a set of declarative rules, that we call physics of the agent environment. On one hand the processes in the environment makes it an active entity which evolves over time without necessarily having agents performing actions in it. On the other hand the services of the environment offers complex functionalities to explore and communicate in the environment.

Figure 1 shows the logical architecture of one GOLEM container. For 3D e-retailing, we have extended our previous work in [4] in four ways. The first is to allow web services to be interpreted as interacting with objects in the agent environment context. The second is to embed an avatar agent representing the user in the environment. The third is to develop a semantic registry containing entities' descriptions, that can be used for discovery purposes, as explained in section 4.2. The fourth is to develop a *positioning system service*, which we explain later in section 4.3.

The user deploys its own avatar in the agent environment by means of a client that connects to a GOLEM container. Through the interface provided by the client, the user can

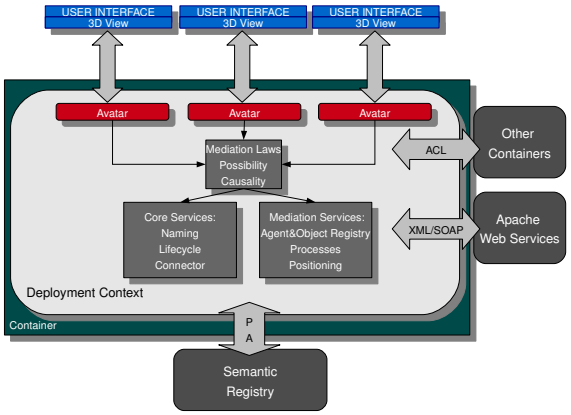

Figure 1: GOLEM Logic Architecture

visualise and interact with the entities populating the virtual environment.

Every container in the GRID communicates with other containers using the messaging facilities. This is done using the *mediation services*, which provide the linking conditions between the different containers, according to the topology of the distributed agent environment. Agents, avatars and objects interact according to declarative rules that govern their interaction, i.e. agents vs. agents, agents vs. objects, objects vs. objects, and agents vs. core/mediation services. This rule-set defines an agent environment physics, coordinating, constraining and enhancing the interaction in a container. The discovery service, that we call *connector service*, provides a link to a *semantic registry* containing the descriptions in WSML (Web Service Modelling Language) [18] of the entities populating the system. The general idea is that such a registry answers to the agents' queries about the position of the other entities of interest, like agents, avatars, objects and containers. The connector provides also the *message core service*, which is an interface to communicate by means of speech acts using an ACL (Agent Communication Language), with entities residing in other containers. Finally the agent and object registry contains the description of agents and objects residing in the container. This can be accessed by agents via PA (Physical Actions).

3.2 Objects in the Virtual Environment

In GOLEM objects are passive-reactive entities encapsulating functionalities that can be exploited by the agents. The GOLEM objects can interact with the environment by means of their triggers and emitters: the triggers are activated by the events happening in the environment, while the emitters produce events in the environment as a reaction to events triggering the bevaviour of the object. Affordances of objects inside GOLEM are ontological descriptions providing to an observer the interface of interaction and the characteristics of the object. In other words, the affordances are used to specify both the interface of the objects in terms of methods and the attributes of the object, as its position, its relations to other objects in the distributed agent environment and its 3D appearance. Such a semantic description

is used by the user client to represent the appeareance of the agent environment, allowing the developer to create its own representation of the virtual environment, taking into account the standardised description of the entities.

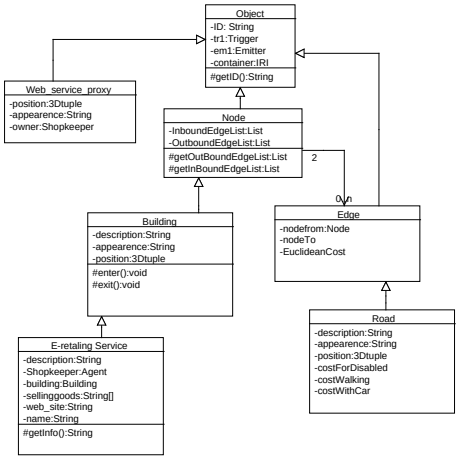

Figure 2: Environment Ontology

In the particular case of the e-retailing application, the objects represent buildings and roads of a virtual environment as well as the content of the shops. Such buildings are logically organised as a distributed convex graph, whose structure is derived by the affordances of the buildings and the roads. The rationale behind this topology is that the object affordances can be used to derive the best path towards a location. Buildings can be both simple scenery or e-ratailing services represented as shops where the user can interact with a shopkeeper agent. The ontology hierarchy representing the objects in the virtual environment is described by the diagram in figure 2. The object affordances are structured as instances of WSML concepts, internally translated in GOLEM to form instances of the environment's state using C-logic [5], see [4] for details. The C-logic description below shows the instance of a e-retailing service:

er_service: s1 [
sellinggoods ⇒ {string:g1, string:g2,string:g3}
methods ⇒ {getInfo, enter, exit}
web_site ⇒ string:ws1,
triggers ⇒ {receptor:r1},
emitters ⇒ {emitter:em1},
appearance ⇒ string:str1,
position ⇒ 3Dtuple:tup1,
outboundEdge ⇒ {Edge:e1,Edge:e2},
inboundEdge ⇒ {Edge:e3, Edge:e4},
description ⇒ string:d1
]

The objects inside the virtual environment that are not buildings or roads, can wrap an interface towards a web-service deployed outside the container providing distributed functionalities in the agent environment. In this case the object works as a proxy to hide the complexity to interface with a web-service, which can be accessed as an object from the virtual environment. In figure 3, two shopkeepers in the agent environment interact with objects representing web services in order to satisfy the user's request.

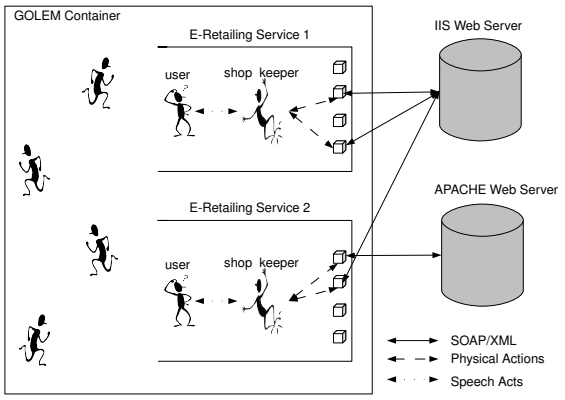

Figure 3: Shop Keeper Agents interacting with objects

The object works as a proxy towards the web service, thus having the same perceivable interface from the point of view of the agent, it is possible to integrate different technologies in the same agent environment.

3.3 Agent Architecture

A 3D client provides a user with an interface to log its own avatar in a GOLEM environment. The avatar is deployed as an agent body with a set of sensors and a set of effectors. On one hand, the client receives the sensors' perceptions and displays them in a view of the 3D environment, according to the semantic descriptions of agents and objects perceived. On the other hand the client can produce actions that are attempted in the agent environment by means of the avatar's effectors. The avatar agent helps the user with the retailing process. It contains a product preference profile based on what the user searches and purchases in order to provide suggestions of interest for the user. The mind of the avatar agent is built following the reactive agent model with preferences over the actions as specified in [17]. Fig. 4 shows the architecture of the avatar and shopkeeper agents, extended from [4].

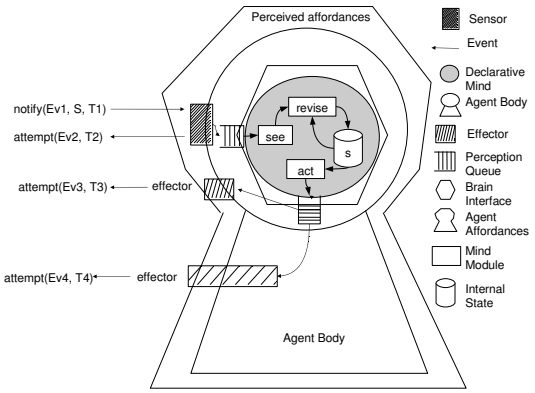

Figure 4: Agent Architecture

GOLEM agents perceive the environment by means of their sensors and effectors, which communicate with a declarative mind through the brain interface implemented in Java. The cycle which explains the agent behaviour is specified in Prolog as follows:

```
cycle(Brain)←
        see(Brain, Percept),
        revise(Percept),
        act(Action),
        execute(Brain,Action),
        cycle(Brain).
```

The above cycle assumes a set of sensors and effectors connected to the brain component. During the see stage, the agent takes the perceptions from the agent sensor, previously notified by the environment about the events occurred in it. The agent sensor has the responsibility to internalise the perceptions which have occured in the agent environment and filter them according to the internal state of the sensor, previously set by the agents (see [4] for further details). The see function selects the sensor from the sensor list and produces a perception in the agent mind taking it that we define in Prolog as follows:

```
see(Brain,Percept)←
        getSensors(Brain,Sensors),
        getPercepts(Sensors,Percept).
```

The above Prolog predicate implements a simple first-in first-out queue that is filled every time a new perceivable event happens in the agent environment. The revision stage takes the perception and the previous state of the agent and maps it to a new internal state. We implement it in Prolog through a set of revise predicates as the example below:

```
revise(do(user,speech_act,Needs))←
    Needs = user_needs(catalogue(A, X)),
    X = item([H|T]),
    update_profile(item([H|T])).
```

The action stage considers two different kind of actions, action on the environment like physical actions and speech acts, and perception acts. The act function below selects the action to perform according to its priority and according to the agent state. After the action is selected, the excute function calls the brain interface to perform a physical act/speech act using the effectors, or to perform a perception act using the sensors.

```
act(Act)←
   findall(select(Label, Act), select(Label, Act), Acts),
   highest_priority(Acts, Act).
```

An example of a selection rule is given below:

```
select(r1,Act)←
   state(user_needs(catalogue(A, item([H|T]))),
   X = speech_act(MyID,A,catalogue(A,item([H|T]))),
   myID(ID),
   Act = act(mouth, do(ID, speech_act, X)).
```

The rule above specifies that the avatar agent executes a speech act towards a shopkeeper agent if the user needs a catalogue of products, where the term item([H|T]) specifies that the user wants a catalogue of item described by the list of words [H|T].

As for the objects, agents are described by means of their affordances. The agent affordances define what is perceivable of the agent, declaring the state of the agent at any time. Such affordances are also used to define the relationship between the e-retailing services and the shopkeeper in charge of them, as well as defining the relationship between the objects that works as an interface towards web services and the agents owning them in the virtual environment.

3.4 Feedbacks between Avatars, Users and Shopkeepers

In order to help the user with the e-retailing process, the avatar agent keeps track of the user interaction. In particular, to achieve this goal the avatar agent considers the queries performed by the user and translates them according to the vector space model for document retrieval as presented in [3]. This idea can be expressed as follows:

$q_1(w_{11}, w_{12}, ..., w_{1X})$
$q_2(w_{21}, w_{22}, ..., w_{2Y})$
$q_K(w_{K1}, w_{K2}, ..., w_{KZ})$

where w_{ij} is the j^{th} word of the i^{th} query q_i. Given the ordered set of words S_{words} which contains all the words that compose the queries, with $N = |S_{words}|$, it is possible to translate every query in a binary vector of length N as follows:

$d_{q1}(t_1, t_2, t_3, ..., t_N)$
$d_{q2}(t_1, t_2, t_3, ..., t_N)$
$d_{qK}(t_1, t_2, t_3, ..., t_N)$

where t_i has value one if the i^{th} of the S_{words} is present in the query, zero otherwise. Then the avatar agent can calculate the *similarities* between two queries as follows:

$q_1(t_{11}, t_{12}, ..., t_{1X})$
$q_2(t_{21}, t_{22}, ..., t_{2Y})$
$d_{q1}(t_1, t_2, t_3, ..., t_N)$
$d_{q2}(t_1, t_2, t_3, ..., t_N)$
$similarity_{(q1,q2)} = cosine(d_{q1}, d_{q2}), \qquad where:$
$cosine(d_{q1}, d_{q2}) = (d_{q1} \cdot d_{q2})/\|d_{q1}\|\|d_{q2}\|$

As a consequence, when a user queries a shopkeeper, the avatar agent can use the similarity definition to "suggest" queries related to the one proposed by the user. For example, let us consider the comparison of two queries of dimensionality 5 as follows:

$q_1('mobile', 'phone', 'nokia', '7410')$
$q_2('mobile', 'phone', 'nokia', '6689')$

translated into the vector space model this would look like:

$d_{q1}(1, 1, 1, 1, 0)$
$d_{q2}(1, 1, 1, 0, 1)$

then applying the $cosine$ similarity we have:

$cosine(d_{q1}, d_{q2}) = \frac{1*1+1*1+1*1+1*0+0*1}{\sqrt{4}*\sqrt{4}} = \frac{3}{4}$

Since the two queries are close in the vector space model, the agent can tell the user that people in the past that have queried for "nokia mobile phone 7410" also have queried for the model "6689". For the shopkeeper agents we use association rules as defined in [1] to identify uncover hidden patterns in data sets, in our particular case of application that is to find relationships or correlations between queries and products. For example, let us say that a shopkeeper agent is queried about a certain product, additionally to the

list of products related to the query the agent can look into the purchase records to obtain additional information. For example (1) avatars that have queried for product "mobile" also have queried for product "head phones" and "case", (2) avatars that have ordered product "mobile" also often have ordered product "car charger" and "case".
This kind of analysis helps introducing feedback from other avatars' experiences, which in turn helps inexpert avatars to explore the information space. It is important to highlight the need to log the queries and purchases in the agent knowledge base in order to carry out the kind of analysis presented above. With this setting when an avatar asks the shopkeeper's catalogue of items related to the description provided by the user, rather than just providing the products that have a description (document) similar to the query proposed by the user, the shopkeeper also recommends products on the basis of the previous interaction with other users.

4 Environment Interaction

The affordances of agents and objects allow us to describe the GOLEM agent environment as a composite and declarative structure that evolves over time. As we did in [4], to capture such an evolution, we define the interaction rules of the agent environment as an extension of the Object Event Calculus (OEC) as defined by Kesim and Sergot in [9], to keep track of the temporal evolution of object and agents in the agent environment. Moreover, to ease the distributed interaction between agents and agent environment we define environment services which offer further informations about the environment, performing calculation about its topology or discovering other entities in it.

4.1 Interaction rules

In order to take place, the actions performed by an agent has to be attempted, and, if possible, they happen in the agent environment and they are translated to OEC events, otherwise the rules of the environment prevent the actions from happening. For example a moving event from one position to another in the 3D environment at time t1 can be specified as

```
happens(e1,t1).
act(e1,move).
actor(e1,ag1).
object(e1, [1,0,1]).
```

where ag1 is the identifier of the agent performing the action. In GOLEM a developer can define what events are possible enumerating all the possible actions, using possible/2 predicates, or defining constraints on the actions using impossible/2 predicates. With respect to the application presented in this paper an example of an impossible rule is the following one:

```
impossible(E, T)←
        do:E [actor ⇒ A,
        object ⇒ Ob],
        instance_of(Ob,web_service_proxy,T),
        not holds_at(A,agent,owns, Ob,T).
```

The rule above states that an agent cannot use a web_service_proxy object if not owned, where holds_at/5 is a predicate defined in OEC to derive the state of a particular object at a particular time T. Since the shopkeeper's objects represent important resources and a direct interaction between the user and the objects may create security issues, we can specify that the user cannot use a certain class of objects, if it is not the owner of them.
We specify similar rules for the interaction between agents, in particular we define rules to limit the physical distance of interaction between avatars and shopkeepers and we define rules to limit the movement of the avatar inside the agent environment, to avoid the compentration of objects and to prevent a user from moving outside the border of the virtual world.

4.2 Connector Service

The connector service offers the basic discovery and message passing facilities for the agent environment. In particular it provides the interface to deliver a message from one container to another. Given a tuple representing a speech act action as below:

```
do(S, speech_act, speech_act(S, R, M))
```

if the receiver R is in a different container to the sender S, the connector delivers transparently to the agent the message to the destination, according to the absolute identifier of the receiver which is a composition of the container ID and of the local ID of the agent. The connector service works also as a proxy towards a semantic registry. In this case an agent producing a physical action in the environment of the kind:

```
do(S, physical_act, query(S, conditions([H|T]))
```

will trigger the interface to query the semantic registry which is deployed in the distributed agent environment as a centralised repository that every container can query. We implemented the semantic registry also in the OEC formalism. The WSML ontologies [18] representing the agent environment are registered in the semantic registry and internalised to a logical description compliant to the OEC. The following first order logic predicates show part of the e-retailing service concept schema:

```
is_a(er_service, building).
attribute(er_service, outboundEdge, multi).
attribute(er_service, description, single).
attribute(er_service, sellinggoods, multi).
attribute(er_service, appereance, single).
```

An instance of this concept is then represented as a set of assign statements in OEC as follows

```
happens(e1,T) .
event(e1).
assign(e1, er_service, sellinggoods, ['mobile','camera']) .
assign(e1, er_service, brand, ['Motorola', 'Philips', 'O2']) .
```

Consequently an agent looking for a mobile phone shop would query the connector interface using these constraints in the query:

now(T),
holds_at(ID,er_service, sellinggoods, L1, T),
member(['mobile'], L1).

The version of OEC we are using is the one optimised in [10] that makes the OEC much more scalable as a formalism for temporal databases.

4.3 Positioning System Service

The agent environment provides the agents and avatars with additional functionalities about the topology of the environment using a *positioning system service* (PSS). The PSS uses objects' affordances to calculate the shortest path from the location of the avatar to another location using an adaptation of the A* algorithm [12] for distributed containers. When a node is in another container, the parameters of the algorithm are sent to the PSS of the second container, which calculates the path on behalf of the starting container. This interaction between two or more containers goes on until the path is found. The final result is then returned to the avatar/agent using the connector core services of the containers involved.
An example of how this process works is given in Fig. 5, showing a set of e-retailing services represented as shops and distributed in several GOLEM containers. An avatar located in container c00 needs to find a specific e-retailing service. The semantic registry provides as a result the services 15, 12, 25. Service 15 is in c20, service 12 is in c11, and service 25 is in c02. The agent decides for the service 12. According to the capabilities and preferences of the avatar, the shortest path to 12 is provided.

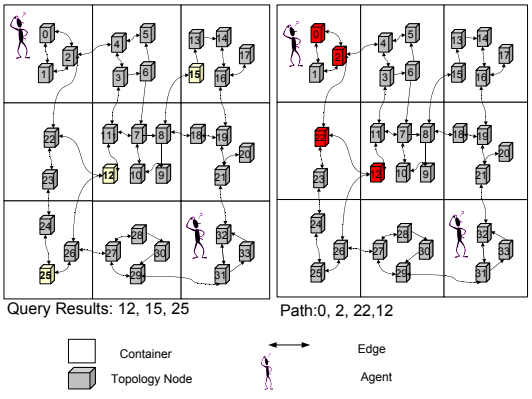

Figure 5: Query and shortest path results

The exchange of messages happening inside the GOLEM framework is exemplified better by the sequence diagram in figure 6.
In figure 6, PSS1 starts to calculate the path between one place to another in the distributed agent environment. At a certain stage the PSS1 discovers a node in the path which does not belong to the container, as a result the PSS1 delegates to the PSS2 (residing in the second container) the calculation of the path submitting the partial result. Notice that the A* algorithm deals with loops already, as a consequence the partial result is not submitted back to PSS1 unless a better path to a node previously visited is found,

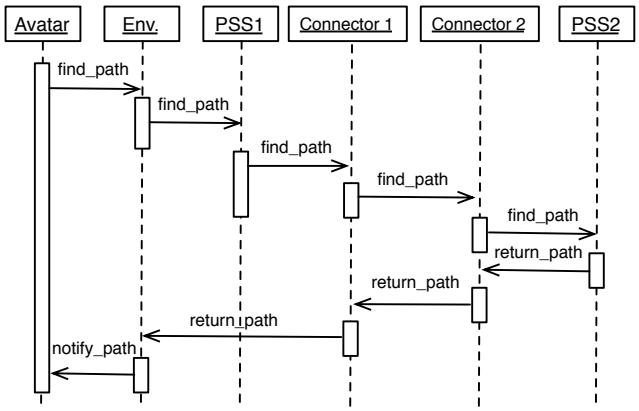

Figure 6: PSS Message Exchange

in that case the PSS2 submits back the partial results. This kind of interaction between the two PSSs goes on until a final result is found and returned back to the user that asked for it. The advantage to embed such functionality in the agent environment, rather than delegating every calculation to the user client, is that the client does not know a priori the whole topology of the distributed environment. In general, embedding such functionalities in the agent environment makes the overall system more flexible. In particular this solution allows us to change the topology of the agent environment at runtime. Changing the topology is as simple as updating the objects' and agents' affordances in one or more containers, without any need to turn off the system, undeploy every entity, update all the clients and then turn it on again. As a result, with respect to Google Maps, creating an A* capable to calculate a path according to multiple criterias is just a matter of adding a parameter to the object affordances representing the convex graph and query the PSSs specifying a different criteria for the cost.

5 Evaluation: Execution Example

The prototype that we have developed, shown in Fig 7, represents a 3D environment that virtualises parts of a metropolitan area.

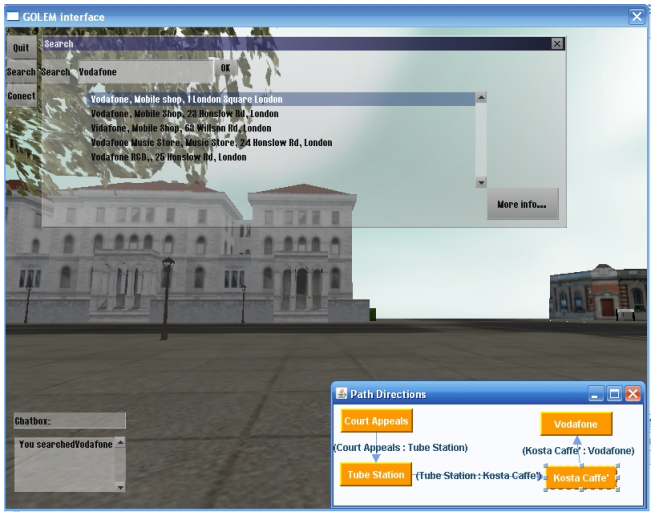

Figure 7: Interface of E-retailing Application

In the context of the resulting virtual world we want to support a user to engage in e-retailing activities by using an *avatar* to explore that world. The world contains a virtual market place of e-retailing services represented as shops, shopkeepers agents and web-services utilised by the shopkeepers. Our prototype assumes that the number of e-retailing services, agents and products available in the virtual environment will be large. In this setting we imagine that the avatar could explore the virtual enviroment making use of the directions and signs he can find in it, but this approach deals only partially with the complexity of services and products proliferation that in a virtual city can be in the order of thousands. Indeed, the avatar would still need to cater for services that are not visible and are accessible only when one enters a building. One way to deal with finding a service is to ask other avatars and agents. Although this could be a reasonable way to proceed for small scale environment where everyone is trusted, it is not as effective as providing search mechanisms that support *path finding* for services as it happens in Google Maps. The top-right window in Fig 7 shows how an agent can use the GOLEM environmental services to find a path for a particular shop.

Experimenting with our prototype requires a set of host computers, each deploying a set of container as a portion of the virtual environment, with buildings, e-retailing services and agents. Buildings are organised as a connected graph and may contain additional structures hosting shops. A shop contains web-services with every service being associated with a shopkeeper agent that interacts with other agents and user avatars. Shopkeeper agents can be in charge of one or more web-services. Once the user has chosen a particular shop, the agent environment can provide directions to reach the destination inside the distributed agent environment. Figure 8 shows an example of an avatar having reached the mobile phone shop.

Figure 8: An avatar entering a virtual shop

Once the user's avatar has reached a shop, it can interact with the shopkeeper agent. The user can ask, for instance, for the product catalogue and the agent can call the web services for the product advertised and return back the list to the user. Once the user has the list, he can purchase a product or close the communication and visit another part of the virtual environment.

6 Related Work

At the initial stages of the project we considered the option to use our system on Second Life [11], a 3D platform enabling interaction between user avatars and 3D objects which can encapsulate remote procedure calls towards external web services. In such system the interaction is embedded in the 3D models to support predefined action rules hard coded for the application. However, we found that the event driven interaction model of Second Life limits the reasoning, resource discovery and cognition about the virtual environment. Our focus has been to maintain a separation between visualisation issues and possible interactions with the virtual environment. This bring the advantage that if the visualisation part changes, the underlying system can remain the same. Moreover our approach has the advantage to use standard ontological descriptions for the entities, where Second Life uses a proprietary languange that makes the integration with other technologies an issue.

Agents for 3D e-retailing has already been discussed in [8], where intelligent communicative agents are developed in JADE [7] to interact with a user in natural language, following interaction protocols based on the user profile and preferences. Although we lack natural language in our prototype, in our system we do not have only agents, but also objects and services that can be discovered dynamically, without reducing interaction to agent communication only. Our framework is richer in that uses the concept of agent environment which allows us to have both very complex agents interacting with the user and a complex distributed environment where the web services are deployed and searchable by mean of their ontological description. As a consequence, there is no need to embed every functionality in the agents, the agent environment can be engineered to provide the support to perceive objects and agents, as well as to act on them physically, rather than limiting the interaction to speech acts.

From the point of view of agent environments in this work there are a lot of connections with the works of Vizzari [14] and Platon et al in [13, 16]. On one hand, Vizzari models the concept of agent environment as a multi-layer multi-agent situated system (MMASS). The environment is composed by a set of graphs interconnected by interfaces, forming a multilayered structure with interfaces amongst layers. Every layer and 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 and affordances. Different sets of rules can then describe different layers of the agent environment and different set of attributes defines the sites occupied by agents and objects in the multilayered structure. The main difference with Vizzari's work is that we model 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. Platon et al. model 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* [13] and *overhearing* [16]. In our work the oversensing/overhearing acts are modeled 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.

7 Conclusions and Future Work

We have presented a prototype multi-agent system whose goal is to support a 3D application for e-retailing. The prototype has illustrated how the use of agent environments can be amongst the most promising and flexible approaches to engineer e-retailing applications. We have shown how the agent environment GOLEM uses semantic web-services to develop the e-retailing application. In this context we have described the features of GOLEM that allow a user to become an avatar and explore the environment that supports agents to find paths and dynamically discover products and services. We also discussed how agents and user interact in the agent environment, profiling the users and trying to satisfy the user needs maximising their profits.
Future work involves adding libraries of objects, services, and avatars, as well as allowing retailers to add content dynamically that will persiste in the virtual environment as active and autonomous entities acting on behalf of the users and retailers. Another direction which will be investigated is the embedding of much clever agents in the agent environment, using one of the architectures proposed in literature as the BDI architecture [17], to simulate an interaction where avatar agents and shopkeeper agents perform negotiation about a product.

Acknowledgments

We would like to thank Servan Keondjian, Jamie Fowlston, Jarred McGinnis and Alex Wrottesley for discussions on the topic of this paper. The work was partially supported by the London Development Agency and the EU IST-6 ArguGRID project.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, April 2004.

[3] Michael W. Berry and Murray Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval (Software, Environments, Tools)*. SIAM, 2005.

[4] S. Bromuri and K. Stathis. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multiagent Systems (EEMMAS'07)*. Springer, Oct 2007.

[5] W. Chen and D. S. Warren. C-logic of Complex Objects. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 369–378, New York, NY, USA, 1989. ACM Press.

[6] J. J. Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, 1979.

[7] JADE. Java Agent DEvelopment framework, 2007. Home Page: `http://jade.tilab.com`.

[8] K. Kamyab, F. Guerin, P. Goulev, and E. Mamdani. Designing agents for a virtual marketplace. *AISB Journal*, 1(1), Oct 2001.

[9] F. Nihan Kesim and Marek Sergot. A Logic Programming Framework for Modeling Temporal Objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):724–741, 1996.

[10] Nihan Kesim. *Temporal Objects in Deductive Databases*. PhD thesis, Imperial College, 1993.

[11] Second Life, 2007. `http://secondlife.com/`.

[12] Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer, 1982.

[13] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. Oversensing with a softbody in the environment - another dimension of observation. In *Proceedings of Modelling Others from Observation'05*, 2005.

[14] Giuseppe Vizzari. *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.

[15] D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.

[16] Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors. *Environments for Multi-Agent Systems II, Second International Workshop, E4MAS 2005, Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Papers*, volume 3830 of *Lecture Notes in Computer Science*. Springer, 2006.

[17] M. Wooldridge. *MultiAgent Systems*. John Wiley and Sons, 2002.

[18] WSMO. Web Service Modelling Ontology, 2008. Home Page: `http://www.wsmo.org/`.