Game-based E-retailing in GOLEM Agent Environments

Stefano Bromuri, Visara Urovi, Kostas Stathis

Department of Computer Science. Royal Holloway, University of London

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 supports social interactions and how it combines them with semantic web technologies to develop the e-retailing application. We also describe the features of GOLEM that allow a user to engage in e-retailing activities in order to explore the virtual social environment by searching and dynamically discovering new agents, products and services.

Key words: agent environments, e-retailing, social interaction, games

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 [20] 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 [33] 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 (Generalized Onto-Logical Environment for Multi-agent systems) [4] as the
agent environment. GOLEM supports the semantic description of services and interactions among user, agents and services.

1.1 Scenario and Motivation

Our e-retailing application is motivated by a scenario where a user consults the application to find/request a product in a virtualised environment of a real metropolitan area (i.e London). We consider a case where a user wants to buy a mobile phone. The user (i.e. a tourist) has a limited knowledge of the metropolitan area, it does not know where the mobile shops are located, nor how to reach them.

We model an e-retailing application that supports users to discover/purchase services and products with a multi agent system situated in a 3D environment which is a reproduction of a real metropolitan area.

There are four goals we want to achieve with this setting.

**Discovery**: the user needs to locate shops that sell a particular brand of mobile phones in the area. At the current state of the art, the user would visit several mobile phone web-sites and search the nearest mobile shop through the provided web page tools. Currently, the World Wide Web organises content in terms of a distributed hypertext accessible via keyword based algorithms. As pointed in [13] the limitations with the current web model are that (i) the increasing content volume makes difficult to locate resources of interest specially in cases where resources contain common words (ii) the non structured web content is hardly accessible to software agents as there is not a standard way for software agents to reason about plain text (iii) opens and integration of the system with new technologies is problematic due to the non standard description of the entities. To create a 3D e-retailing agent environment that is open and at the same time accessible by both human an agents for reasoning and discovery purposes, requires semantic annotations for the entities populating it, in such a way that the entities populating the environment can issue structured queries to it in order to know the whereabouts and the features of a particular entity.

**Best route**: the user searches the best route to visit a shop. In our 3D e-retailing prototype, we want have an agent environment that is a reasonable reproduction of a real metropolitan area. As a consequence having a topology associated to the environment that can be navigated, helps in accessing the content deployed in the application as well as in having a better knowledge of the associated real environment. We investigate the use of semantic annotations to allow a complex interconnected topology to be navigated as a graph in such a way that users and agents populating it can discover the best route.
to a interesting and previously discovered location just by issuing queries to the agent environment.

**Personalisation and Interaction**: provide profiling capabilities for the user to improve the browsing experience by providing useful suggestions on products of interest. Such an interaction is generally based on a simple and ad hoc purchase protocol, where the agents have no memory about the user, nor the user is guaranteed about the success of the transaction, because no authority component is envisaged. In this case we are motivated by the issue of modelling the transactions happening between a retailer, that we call shopkeeper agent, and a user, that we call avatar, as well as modelling the result of this transaction in terms of a contract between the shopkeeper and the avatar.

**Persitence**: the user persists in the virtual word also if it is off-line. The current approach is that when the user is off-line also his avatar is not present. A 3D e-retailing application could benefit from an avatar that acts on behalf of the user when the user is off-line, behaving as an active and persistent avatar in the virtual environment.

### 1.2 Contribution

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 technologies 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 [36], 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.

Unlike our earlier work in [5], we model the agent environment as a physical environment that facilitates a higher-level social environment. The physical environment specifies what physical events are possible (or impossible) to occur in it, acting as a virtual physics for the entities situated in it. The social environment on the other hand defines the rules of the social interaction characterising an e-retailing application. We model the social environment using
the games metaphor [31]. Under this metaphor, agent interactions of an e-retailing application count as moves of a complex game composed of simpler, possibly atomic sub-games, governing the interaction between the participating agents that enact the roles of players. The outcomes of these games will not necessarily give rise to a winner or looser, but more likely to win-win situations where a number of agents achieve their goals, such as a situation where an agreed sale results in mobile phone contract.

We will demonstrate how to control complex (or compound, as we like to call them) games using an object called the Games Calculator. This object is used as a coordination device that is shared by agents to check the validity of moves between the parties involved in the transaction and validate the results, for instance, a valid contract resulting from the agreements between the parties. The resulting framework is an agent environment where agents are organised in terms of complex and social multi-party games to buy or sell items, with the possibility to enforce the rules of the interaction if a party has not respected them. We will argue at the end that enforcing the rules depends on how the designer of the social environment organises objects and the roles of agents in it for a specific application.

1.3 Organisation

The remainder of this paper is organised as follows. In section 2 we discuss the main GOLEM platform that will support e-retailing services, agents, objects and the 3D visualisation envisaged. In section 3 we describe how the physical interaction takes place inside the agent environment supported by GOLEM. Then in section 4 we show how build on top of GOLEM a social interaction framework based on games to support interactions amongst agents in the environment. To illustrate how a user interacts with the e-retailing application, we present a possible virtual environment execution in section 5. How the work relates to existing literature is presented in section 6 and, finally, section 7 summarises our contribution and discusses our plans for future work.

2 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 multi-agent 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 [12], 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.

2.1 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 constrains 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 [4,5] in five 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 3.2. The fourth is to develop a positioning system service, which we explain later in section 3.3. The fifth is to develop a complex game mediating service which works as a social environment mediating the negotiation in our e-retailing application.

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 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 the agent environment physics and the agent environment social law, 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 Modeling Language) [37] 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).

2.2 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 behavior 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 appearance 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.

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-retailing 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 [6], 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.

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
2.3 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 [35]. Fig. 4 shows the architecture of the avatar and shopkeeper agents, extended from [4].

GOLEM agents perceive the environment by means of their sensors and effec-

Fig. 3. Shop Keeper Agents interacting with objects integrate different technologies in the same agent environment.

Fig. 4. Agent Architecture
tors, 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),
    choose(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 occurred 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])).

As specified in [4], the agent can internalise the declarative description specified as affordances within its mind via its sensors in order to reason about the dynamic evolving environment in which it is situated. Different agents may have different views at a given time of the same agent environment. If the agent environment was just a message passing system, this would lead to many inconsistency issues, such as agents that try to perform actions on objects that are not available or accessible any more. In the case of GOLEM, as we will illustrate in section 3, the agent environment works as a mediator for the interaction between agents and objects, as a consequence, if an agent tries to perform an action which is impossible according to the rules of the agent environment, such action is going to be prevented from happening.
The choice stage considers two different kind of actions, action on the environment like physical actions and speech acts, and perception acts. The \texttt{choose} function below selects the action to perform according to its priority and according to the agent state. After the action is selected, the \texttt{execute} function calls the brain interface to perform a physical act/speech act using the effectors, or to perform a perception act using the sensors.

\begin{verbatim}
choose(Act)←
    findall(select(Label, Act), select(Label, Act), Acts),
    highest_priority(Acts, Act).
\end{verbatim}

An example of a selection rule is given below:

\begin{verbatim}
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)).
\end{verbatim}

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 \texttt{item([H|T])} specifies that the user wants a catalogue of item described by the list of words [H|T]. In particular the interaction between the avatar and the shopkeeper agent triggers the social environment to start a compound game for the purchase of an item or a set of items as we will see in section 4.

The agent mind can work as a coroutine with the agent body: this means that the agent mind may provide multiple actions at each round for the body. Such actions will be taken from the effector queue (see figure 4) and executed in parallel at each round, but only in the case that they are addressed to different effectors of the agent. In other words it is perfectly possible for the agent to perform the action \texttt{walk} while performing the action of \texttt{speak} but only in the case that they are complementary actions and that they are conform with the physics of the agent environment.

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 The Physical Environment

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 [17], 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 information about the environment, performing calculation about its topology or discovering other entities in it. In particular, the component in charge to mediated the interaction happening between the agents within the environment is the Mediation Laws system component, that is specified as a declarative component built around the OEC specification (see [4]).

3.1 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

\[
\text{happens(e1,t1).}
\text{act(e1,move).}
\text{actor(e1,ag1).}
\text{object(e1, [1,0,1]).}
\]

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 \text{possible/2} predicates, or defining constraints on the actions using \text{impossible/2} predicates. With respect to the application presented in this paper an example of an impossible rule is the following one:

\[
\text{impossible(E, T)←}
\text{do:E [actor ⇒ A,}
\text{object ⇒ Ob],}
\text{instance_of(Ob,web_service_proxy,T),}
\text{not holds_at(A, owns, Ob,T).}
\]

The rule above states that an agent cannot use a web_service_proxy object if not owned, where \text{holds_at/4} 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 penetration of objects and to prevent a user from moving outside the border of the virtual world.

3.2 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:

\[ \text{do} (S, \text{speech\_act}, \text{speech\_act}(S, R, M)) \]

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.

Another function of the connector service it to work as a proxy towards a semantic registry. Querying the connector service the agents can discover the location of e-retailing services or agents selling a particular good of interest.

In this case an agent producing a physical action in the environment of the kind:

\[ \text{do} (S, \text{physical\_act}, \text{query}(S, \text{conditions}([H|T]))) \]

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. In other words our registry works as a yellow page component for our system. We implemented the semantic registry also in the OEC formalism. The WSML ontologies [37] 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:

\[ \text{is\_a(}\text{er\_service}, \text{building}) \]
\[ \text{attribute(}\text{er\_service}, \text{outboundEdge}, \text{multi}) \]
\[ \text{attribute(}\text{er\_service}, \text{description}, \text{single}) \]
\[ \text{attribute(}\text{er\_service}, \text{sellinggoods}, \text{multi}) \]
attribute(er_service, appereance, single).

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

\begin{verbatim}
  happens(e1,T).
  event(e1).
  assign(e1, er_service, sellinggoods, ['mobile','camera']) .
  assign(e1, er_service, brand, ['Motorola', 'Philips', 'O2']) .
\end{verbatim}

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

\begin{verbatim}
  now(T),
  holds_at(ID,er_service, sellinggoods, L1, T),
  member('mobile', L1).
\end{verbatim}

The version of OEC we are using is the one optimised in [18] that makes the OEC much more scalable as a formalism for temporal databases.

### 3.3 Positioning System Service

After querying the connector service about the location of the e-retailing services in the agent environment, the user may want to discover the best route from its actual position to the place of interest. 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 the location previously discovered using an adaptation of the A* algorithm [22] 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.

The exchange of messages happening inside the GOLEM framework is exem-
plified 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. 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, thus making the overall system more flexible. For instance, we can change the topology of the agent environment at runtime, without having to restart the system and redeploy every entity and clients of the application. More details about the advantages of our approach can be found in [5].

4 The Social Environment

We use the games metaphor [31,29,30] for formulating the social interactions amongst agents. This allow us to define the social rules governing an agent environment as the rules of a complex game. In this context, the state of the game describes the state of the social interaction. Agents are players whose
roles in the game are the roles that agents play in an application. The legal moves of a player in a game are events that are possible in the agent environment and compliant to the social rules of the protocols used for an application. Specifying the legal moves of a game corresponds to defining the preconditions of actions in the social environment. The rules of a game also describe how the moves affect the game’s state representing the social state of the application.

Interactions formulated as a game are regulated and produce a result. The result does not necessarily need to be zero-sum [21], by requiring a winner and a loser, but it can also give rise to a win/win or loose/loose situation, depending on the application and the multi-agent system at hand. One major advantage of modeling the social interaction as a game, is that we support negotiating situations where a user is trying to find good deals according to a set of preferences quite naturally. A games approach also determines without ambiguity an agent’s commitments in a negotiation process, as this must be specified in the interaction rules.

We will use the notion of atomic games to define simple agent protocols. We model more complex interactions as a complex game composed of sub-games as discussed in [29]. More complex games correspond to more complex social activities between agents, such as workflows, allowing for different possible interactions between participants that can produce complex outcomes. We assume that games come with an ontology that agents need to agree before interacting. We do not make any assumption on agent’s strategies, preferences or intents, but we expect there would be a preference relation as with the definition of priorities in section 2.3. In figure 7 we show how the architecture of the social environment can be embedded in a GOLEM container.
4.1 Contract Games in E-Retailing

E-retailing requires agents to follow protocols, reach agreements and be participants in contracts about products. In order to support these interactions about a product (set of products), multiple negotiations may occur. We model such an interaction using the concept of complex or compound game.

![Diagram](image)

**Figure 8. The Contract Game**

Figure 8 shows the contract game formulated as a compound game with subgames illustrating how to produce a contract for a mobile phone. Agents first select a role negotiation game in order to agree a role in the interaction that follows (e.g. buyer and seller). Once the roles are agreed, the agents can select one or more term negotiation games depending on the number of product attributes the agent wants to negotiate about. Terms in this context include the tariff, the device, monthly payments, if any, and other important information for the contract. Once these terms are established, the signature game concludes the contract, with the agents giving their consensus about the terms agreed in the contract game. In other words, the result of the compound contract game is an agreement specified in a contract between buyer and seller.

4.2 Atomic Games for Contract Negotiation

In this section we focus on the rules that define the evolution of a term negotiation game. As with GOLEM actions, we specify the moves in a game in terms of GOLEM events, expressed as C-Logic terms:

\[
\text{do:ev1[actor } \Rightarrow a1, \text{ act } \Rightarrow \text{ accept, role } \Rightarrow \text{ buyer }]
\]

A game state is also described by a C-Logic complex term. Creation of the state
is treated as a complex property initiated by an object-based event calculus initiates/4 definition. The C-Logic terms below shows how an instance of an atomic game such as the term negotiation game is represented in our approach.

term_negotiation:tn1[
  agents⇒{agent:a1[role⇒seller], agent:a2[role⇒buyer]},
  requested⇒(mobile_phone:m1[id⇒"mpid1",color⇒c1, price⇒200],
  proposed⇒(mobile_phone:m2[id⇒"mpid1", color⇒c1, price⇒50],
    tariff:t1[price⇒10,period⇒month, duration⇒12]),
  last_move⇒counter_propose:ev256)].

A term negotiation game has a set of agents playing roles, a set of requested products, a set of proposed products and a the last move being made. The moves consist of five utterances of request, propose, counter_propose, accept, reject locutions. Role negotiation and the signatures atomic games are represented similarly. At any time, we link a physical event to a social term by specifying the legal moves of a game by means of legal_at/3 predicates, as follows:

legal_at(Game, Move, T) ←
  instance_of(Game, term_negotiation, T),
  do:Move[actor⇒A, act⇒request, role⇒buyer],
  holds_at(Game, agent_of, A, T),
  holds_at(A, role, buyer, T).

This rule means that it is legal to make a request move in a term negotiation game only if the agent is indeed the buyer. We express the evolution of the game in terms of initiates/4 and terminates/4 predicates, according to the events happening in the games. For example the rule specification:

initiates(Ev, Game, requested, Product)←
  happens(Ev,T),
  Ev[act⇒request, item⇒Product],
  instance_of(Game, term_negotiation, T).

initiates a requested product to be stored in the state of the game as a result of a request being made in a term negotiation game. An old request is terminated and substituted by a new request because of the way the object event calculus is specified (for the technical detail of this the interested reader is referred to [4], section 3.1, axiom C9).

The evolution of a game term will eventually terminate. We specify the terminating conditions of a game through the terminating_at/2 predicate. For example, the rule:

terminating_at(Game, T)←
  instance_of(Game, term_negotiation, T),
  holds_at(Game, last_move, E, T),
E[act ⇒ accept].

specifies that the result of a term negotiation game ends when the request of a buyer is matched by the proposal of a seller. Attributes of the state of a terminated game can then be accessed by querying the state term using the version of C-Logic specified here.

4.3 Contract Negotiation as a Compound Game

Atomic games in our e-retailing environment exist in order to write a contract about the negotiated products. We represent the contract negotiation as a compound game, which, as with atomic games, is specified in term of legal.at/3, initiates/4, terminates/4, and terminating.at/2 rules. An instance of a contract game is shown below:

```
contract:g1[
  header ⇒ {
    (agent:a1,role:r1),(agent:a2,role:r2)
  },
  parts ⇒ {
    role:negotiation:rn1, terms:negotiation:tn1, signature:s1
  },
  agreed:terms ⇒ {
    (mobile:phone:m1[price ⇒ 80, model ⇒ m1],
    tariff:t1[price ⇒ 10, period ⇒ month, duration ⇒ 12]),
    case:c1[price ⇒ 10, color ⇒ black]
  },
  signatures ⇒ {s1,s2}
].
```

A contract has a header stating which are the participants and what is their role in the contract, a parts attribute representing the sub-games required to establish the agreed terms of the contract and the signatures of the participants. Being a compound game, a contract game needs to represent legal moves at two levels:

- the compound game, for example

  legal_at(Game, Move, T) ←
  instance_of(Game, contract, T),
  do:Move[actor ⇒ A, act⇒start, game⇒role_negotiation, parties⇒Parties],
  not holds_at(Game, header, _, T).

  expressing that if the header attribute has no value in the state of the contract, then it is legal to start role negotiation game to instantiate the header;

- the sub-games, defined generally as

  legal_at(Game, Move, T) ←
  active_at(Game, SubGame, T),
  legal_at(SubGame, Move, T).

18
and stating that a legal move in a compound game is any legal move of its component sub-games.

There are other moves at the compound game level that we have specified for a contract game, in particular, suspending a sub-game, resuming a sub-game, interrupting a sub-game, together with their associated initiates/4 definitions. However, the details of these specifications we omit as they are straightforward. Instead, we show next how we define when a sub-game is active in a compound game. For our contract game above, this is defined as a component sub-game that has not yet been terminated:

\[
\text{active}_\text{at}(\text{Game}, \text{SubGame}, T) \leftarrow \\
\quad \text{holds}_\text{at}(\text{Game, part_of, SubGame, T}), \\
\quad \text{not terminating}_\text{at}(\text{SubGame, T}).
\]

The definition above assumes that the sequencing of the sub-games in a contract game occurs in the definition of legal moves at the compound game level, i.e. we specify that it is legal to start first a role negotiation game that becomes active and then terminates, then we start a series of term negotiation games, that each one of them becomes active and then terminates, and so on, until the contract has been signed. Again the technical details we omit as they are straightforward.

We are now in a position to show in the next section how we implement compound games in order to support the social environment of our e-retailing application.

### 4.4 Implementation Issues

The compound game metaphor we considered can be implemented in different ways. In particular one way is to implement in every agent a representation of the rules and the state of a game, in a completely distributed model, but the coordination rules are duplicated and the agent mind becomes more difficult to program. A second way is to consider an umpire agent to do the checking instead of the players, the advantage being that that the umpire is like a trusted third party doing the validation of the interactions so that the players concentrate on their strategies. A third way is to have a game calculator object that acts as a coordination device between agents, it is both an object that agents need to coordinate in order to interact with it and at the same time it can be a trusted device that validates moves. Our choice is for the third one, since it is a more economic choice and it allows to have agents focusing on their strategies according to their roles.
In figure 9 we show the relationship between the physical environment and the social one, through the use of the Game Calculator service. The agents use the Game Calculator as they use any other service of the GOLEM infrastructure, calling a method of the service by means of actions performed in the environment. The content of such actions represents a move in the compound game.

To implement the compound game metaphor, we link the internal part of the Game Calculator Object with a TuCSoN [25] tuple centre [23], a Linda-like [11] extension of the concept of tuple space as a reactive logic based blackboard. Notice that the agents do not interact directly with the Tuple Centre, whose complexity is hidden by the Game Calculator Service via the processor component, and as a consequence we could translate our game to any other Linda-like system, like LIME [26].

Using this infrastructure the compound game and the atomic games are conceived as a combination of the ReSPeCT language [24] and the Object Event Calculus. In figure 9 the compound game is represented as a tuple centre that communicates with multiple tuple centres representing the sub-games. The reaction shows how we can produce a move in terms of tuples, in the compound game after checking the validity of the action using the legal_at/3 predicate previously defined. By means of these reactions, the agents producing actions on the game calculator through the physical environment, can produce moves in multiple distributed games as well as inspecting the internal state of the Game Calculator. There are many different reactions that we use in the implementation of the game, we omit them because they are out of the scope of this paper.
5 Evaluation

The prototype that we have developed, shown in Fig 10, represents a 3D environment that virtualises parts of a metropolitan area. 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 environment 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.

![Interface of E-Retailing Application](image1)

![A Virtual Shop](image2)

Fig. 10. Interface of E-retailing Application

The top-right window in Fig 10(a) shows how an agent can use the GOLEM environmental services to find a a particular shop communicating with the Semantic Registry explained in section 3.2. Figure 10(b) shows the path to the desired shop found using the PSS service.

The purpose of having PSSs services distributed in every container is to leave the responsibility of a portion of topology to the container in charge of it rather than assigning every responsibility to the centralised yellow pages represented by the semantic registry. One of the reasons for this choice resides mainly in
the fact that a centralised registry could become a bottleneck for the system if charged with many functionalities so it is a good idea to minimise the responsibility of such a centralised component.

On the other hand the topology of our system is distributed in many containers, leaving it to a centralised component limits the dynamicity and robustness of the solution, since for every change of the environment also the registry would have to be changed and since everything in the system would depend on the centralised component.

In order to test the scalability of the PSSs approach, we created a simulation based on the Jung library for graphs [1] for two distinct topologies shown in figure 11. To test the A* in a distributed setting, we considered the main time delays added to a normal A* as the delay to serialise and deserialise the context of the A* and the delay to send the context. The first kind of delays
are more important because in an unfortunate topology the A* may contain a lot of nodes and the file sent could grow quite dramatically. For the second kind of delays we assume a low latency grid network for the containers as in modern game infrastructures, which have and average latency in the order of 0.1-0.3 ms.

The A* is both admissible and optimal when the heuristic function chosen is a reasonable estimation of the cost to reach the destination, as a consequence the number of messages exchanged between the containers is, in the worst case, equal to the number of connections between the two. Running our tests we discovered that the worst case possible is the case when the topology is randomly generated as in figure 11(a). In such a topology the degree of connectivity between the nodes assigned to a container and the nodes assigned to other containers is about 200. In this case, when looking for a result, the PSSs will send the context back and forth and the dimension of the serialized context will grow and consequently also the time to serialize it. The performances for this case are shown by the curve test 1 in figure 11(c), for graphs with a progressive number of nodes, from a very sparse graph to a very dense one, and each degree of incidence for every node equal to 4, where for around 1000 nodes the time to retrieve a result is already around 3s where the graph is distributed on a grid of 25 containers.

Observing the worst possible case, we can make some observations: in order to avoid the bottleneck effect, the developer of the virtual environment could approximate the topology limiting the number of connections between containers.

In figure 11(b) we show a topology of about 1350 nodes, where the average degree of incidence per node is still 4 but the average degree of incidence per container is 5, where every container is connected only with its adjacent neighbours. The performances with topologies of this kind, from very sparse graphs to very dense graphs, is shown by the curve test 2 in figure 11, where for a graph of 1350 nodes we get around 250ms, where the graph is distributed on a grid of 25 containers.

Another consideration that it is necessary to make is about the number of containers needed to map an area. In general there is a trade-off to consider: it is true that more containers allow for more users in the distributed virtual agent environment, but to a larger number of containers corresponds a larger number of changes of context when looking for a path. Considering that every container can support up to 50 users at a time, it is a responsibility of the developer to model its virtual agent environment in such a way that the changes of context are minimized. Further observation can be made about the definition of hierarchical graphs, where containers are developed within containers, to further improve the path discovery, but this is an issue we will consider for
future work. Finally it is necessary to talk about the scalability of the centralised semantic registry presented in section 3.2. A centralised registry to discover the entities of a distributed system is a sensible component because it can work as a bottleneck for the system. Such a registry in our system is implemented on top of the OEC specification which has been already proven to be scalable in [19] for many thousands of entities registered. As a consequence such a centralised registry is suitable for the registration of the description of the e-retailing services, that is duplicated in the containers and in the registry, while the state of the other entities is kept only in the container where they are situated.

6 Related Work

At the initial stages of the project we considered the option to use our system on Second Life [20], 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 language that makes the integration with other technologies an issue. Another advantage of our framework with respect to Second Life is that the negotiation between the user and the retailers is mediated by the social environment, giving us the possibility to model safe transactions.

Agents for 3D e-retailing has already been discussed in [16], where intelligent communicative agents are developed in JADE [14] 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
From the point of view of agent environments there are a lot of connections with the works of Vizzari [32] and Platon et al in [27,34]. Vizzari models the concept of agent environment as a topology representing a multi-layer multi-agent situated system (MMASS) where Platon et al. use the notion of an agent soft body with a public and observable state. The act of observing such a state in the Platon et al. framework is based on the notion of oversensing [27] and overhearing [34]. The similarities and differences with these works have already been discussed in [4,5], the interested reader can look at these works for further details.

From the point of view of social environment, there is a growing research in how to model norm-governed systems. Artikis et al [3] propose a model for norm-governed multi-agent systems as executable specification of open agent societies. This work represents social constraints by making a clear distinction between physical capabilities, institutional power, permissions and sanctions to enforce policies. An agent enters in a society after establishing its role. In this way, the constrains prescribe the behaviour of an agent occupying a specific role. Our work is related to Artikis et al in the sense that we both try to specify social constraints, however, we are more concerned with specifying structure in the social environment in terms of games while their work is more on how to represent normative concepts in rules and proving properties of the overall system. In this respect our approaches are complementary. We are also both dealing with computational aspects of checking social constraints but we use different dialects of the event calculus.

The Electronic Institution (EI) approach [10,8,9] takes a societal and mediated approach to multi-agent systems. EIs are defined in as composed of four main elements: dialogical framework, scenes, performative structure and norms. In most multi-agent systems every agent immersed in the agent environment has its own internal ontologies and languages, but in order to be able to interact with other agents, every agent must share a dialogical framework. The activities inside a multi-agent environment are seen in terms of multiple, concurrent dialogical activities.

Our framework is similar to EIs in the sense that we can represent the dialogical framework as a game in the social environment. We can support internal ontologies and strategies inside the agent mind and multiple games represent multiple scenes where the norms are represented as legal rules. In EIs concurrency is obtained by combination of multiple dialogues: in the same way we can represent multiple active subgames in complex games played concurrently by agents. Finally, EIs abstract away from the result of the dialogical interaction between the agents, while we use such interaction to produce complex structures representing the outcome of the interaction.
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 by means of the social environment, represented as a complex compound game played by the agents to define contracts about the item purchased.

Future work involves adding libraries of objects, services, and avatars, as well as allowing retailers to add content dynamically that will persist in the virtual environment as active and autonomous entities acting on behalf of the users and retailers. Another direction which will be investigated is embedding additional “intelligence” to agents using architectures such as BDI [28] or KGP [15], to support avatars and shopkeeper agents to have their own strategies and preferences to negotiate about a product. We also envisage to use a minimal-reward based strategy [7] as a negotiation strategy between agents.

References


