Using Semantic Web technology in Multi-Agent Systems: a case study in the TAGA trading agent environment

Youyong Zou, Tim Finin, Li Ding, Harry Chen, Rong Pan
U.Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD, 21250
410-455-3971

{yzou1,finin,dingli1,hchen4 }@cs.umbc.edu

ABSTRACT

Travel Agent Game in Agentcities (TAGA) is the framework that extends and enhances the Trading Agent Competition (TAC) scenario to work in Agentcities, an open multi agent environment based on FIPA compliant platforms. TAGA uses the semantic web languages and tools (RDF and OWL) to specify and publish the underlying common ontologies; as a content language within the FIPA ACL messages; as the basis for agent knowledge bases via XSB-based reasoning tools; to describe and reason about services. TAGA extends the FIPA protocols to support open market auctions and enriches the Agentcities with auction services. The introducing of the semantic web languages improves the interoperability among agents. TAGA is intended as a platform for research in multi-agent systems, the semantic web and automated trading in dynamic markets as well as a self-contained application for teaching and experimentation with these technologies.

Keywords

Agentcities, FIPA, Multi Agent System, OWL, Semantic Web, Trading Agent Competition.

1. INTRODUCTION

The Trading Agent Competition (TAC) [Wellman, 2002] was a test bed for intelligent software agents that interact through simultaneous auctions to obtain services for customers. The trading agents operated within the travel market scenario, buying and selling goods to best serve their given travel clients. TAC was designed to promote and encourage research in markets involving auction and autonomous trading agents and had proven to be successful after three consecutive year's competitions.

Although TAC's framework, infrastructure and game rules had evolved over the past three competitions [Stone, 2000] [Greenwald, 2001] [Wellman, 2002], the assumptions and approach of TAC limited its usefulness as a realistic test bed for agent based automated commerce. TAC used centralized market server as the sole mechanism for service discovery, communication, coordination, commitment, and control among the participating software agents. The trading agents communicate with the central auction server through network socket interface, exchanging pre-defined XML-based messages. In real world, the auction servers (for example, priceline.com and hotwire.com) and service providers are distributed among the massive open Internet and have distinct service descriptions and diverse service access interfaces. The abstractness and simplicity of the TAC approach helped to launch it as a research vehicle for studying bidding strategies, but are now perceived as a limiting factor for exploring the wide range of issues inherent in automated trading in open environment.

Agentcities [Dale, 2002] is the international initiative designed to explore the commercial and research potential of agent-based applications by constructing an open distributed network of platforms to host diverse agents and services. The ultimate goal is to enable the dynamic, intelligent and autonomous composition of services to achieve user and business tasks, therefore creating compound services to address changing needs. In such an open and distributed environment, the need of standard mechanisms and specifications is crucial for ensuring interoperability of distinct systems. The Foundation for Intelligent Physical gents (FIPA) produces such standards for heterogeneous and interacting agents and agent-based systems. In the production of these standards, FIPA promotes the technologies and interoperability specifications that facilitate the end-to-end inter-working of intelligent agent systems in modern commercial and industrial settings.

Inspired by TAC, we have developed Travel Agent Game in Agentcities (TAGA) on the foundation of FIPA technology and the Agentcities infrastructure. The agents and services use FIPA supported languages, protocols and service interfaces to create the travel market framework and provide stable communication environment where messages expressed in semantic languages can be exchanged. The travel market is the combination of auctions and varying markets including service registries, service brokerage, wholesalers, peer-to-peer transactions, bilateral negotiation, etc. This provides a richer test bed for experimenting with agents and web services as well as a interesting scenario to test and challenge agent technology. TAGA is running as a continuous open game at http://taga.umbc.edu/ and source code is available for research and teaching purposes.

The next section introduces the TAGA game and six types of agents. The details of using semantic web technology are presented in Section three. We discuss TAGA's features and our research contributions in Section four and suggest the future works in Section five.

2. TAGA GAME AND AGENTS

We design TAGA as a general framework for running agent-based market simulations and games. Our first use of TAGA has been to build a travel competition along the lines that used in the last three year's TACs. In the competition, *customers* travel from City A to City B and spend several days before flying back. A *travel package* includes a round-trip flight ticket, corresponding hotel accommodation and tickets to entertainment events. A *travel agent* (an entrant to the game) competes with other travel agents in making contracts with customers and purchasing the limited travel services from the *Travel Service Agents*. Customer selects the travel agent with best travel itinerary. The objective of the travel agent is to acquire more customers, fulfill the customer's travel package, and maximize the profit.

TAGA provides a flexible framework to run the travel market game. Figure 1 show the structure of TAGA. The collaboration and competition among six types of agents who play different market roles simulate the real world travel market. We find that basing our implementation on FIPA compliant agent platforms has made the framework extremely flexible. We'll briefly describe the different agents in our initial TAGA game.

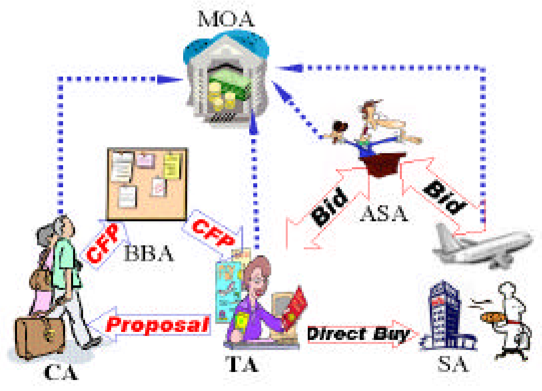

Figure 1: TAGA Architecture

The *Auction Service Agent* (ASA) operates all of the auctions in TAGA. Supported auction types include English and Dutch auctions as well as other dynamic markets similar to Priceline.com and Hotwire.com.

A *Service* Agent (SA) offers travel related service units such as airline tickets, lodging and entertainment tickets. Each class of travel related service has multiple providers with different service quality level and with limited service units. It allows other agents to query its description (e.g. service type, service quality, location) and its inventory (the availability or price of a certain type of service unit). Other agents may directly buy the service units through published service interface. SA also bids intentionally in the auctions to sell its good, e.g. listing its goods in auction and wait for the proper buyer.

A *Travel Agent* (TA) is a business that helps customers acquire travel service units and organize travel plan. The units can be bought either directly from the service agents, or through an auction server.

A *Bulletin Board Agent* (BBA) provides a mechanism helping customer agents find and engage one or more travel agents.

A *Customer Agent* (CA) represents an individual customer who has particular travel constraints and preferences. Its goal is to engage one or more TAs, negotiate with them over travel packages, and select one TA that is able to acquire all needed travel service units.

The *Market Oversight Agent* monitors the game and updates the financial model after each reported transaction and finally announces the winning TA when the game is over.

The basic cycle of the TAGA game has the following five stages:

- A customer-generating agent creates a new customer with particular travel constraints and preferences chosen from a certain distribution.
- The CA sends the customer's travel constraints and preferences to the BBA in the form of a CFP (call for proposal) message. The BBA forwards the CA's CFP message to each of the TAs that has registered with it. Each TA considers the CA's CFP independently and decides whether and how to respond.
- When deciding to propose a travel package, The TA contacts the necessary ASAs and SAs and assembles a travel itinerary. Note that the TA is free to implement a complex strategy using both aggregate markets (ASAs) as well as direct negotiation with SAs. The proposal to the CA includes the travel itinerary, a set of travel units, the total price and the penalty to be suffered by the TA if it is fail to complete the transaction.
- The CA negotiates with the TAs ultimately selecting one from which to purchase an itinerary based on its constraints, preferences and purchasing strategy (which might, for example, depend on a TA's reputation).
- Once the TA has a commitment from the CA, it attempts to purchase the units in the itinerary from the ASAs and SAs. There are two possible outcomes: the TA acquires the units and completes the transaction resulting in a satisfied CA and a profit or loss for the TA, or the TA is unable or unwilling to purchase all of the units, resulting in an aborted transaction and the invocation of the penalty (which can involve both a monetary and a reputation component).

3. AGENT COMMUNICATION

3.1 Agent Communication Model

The previous TACs used a straightforward client-server architecture in which a single TAC server managed all travel service suppliers as well as the customers. Game participants wrote travel agency (TA) agents that connected as clients to the central TAC server. Moreover, these TA agents could only interact with service providers through centralized auction markets. While this architecture greatly simplifies both the development of the TAC infrastructure and the programming of a TAC client, it is a poor model for commerce in the real world.

Peer-to-peer or multi-agent systems offer a more realistic model where customers, service providers and various kinds of "middlemen", including market providers, operate as autonomous peer agents. Moreover, agents can develop complex strategies, which involved a combination of direct transactions (e.g., TA buy direct from hotel agent) as well as auction-mediated transactions of various kinds. Finally, adopting a multi-agent systems approach integrated all aspects of commerce (service discovery, information seeking, negotiation, decision making, commitment, transaction execution et.) in a more natural manner.

The FIPA standards offer mature specifications for multi-agent systems communication, interactions and infrastructure with an emphasis on agent communication languages (ACLs) and protocols. We found the FIPA framework to be a good one for TAGA when augmented with the semantic web languages RDF [zou, 2003] and OWL. In the remainder of this section we will describe the choices made for the content languages.

3.2 OWL as Content Language

The content language is a language used to express the content of messages exchanged between agents. The FIPA communication infrastructure allows agents to communicate using any mutually understandable content language as long as it satisfies a few minimal criteria as a FIPA compliant content language. Published FIPA specifications provide a library of registered FIPA compliant content language, including FIPA-SL, XML and RDF. A good content language should be able to express rich forms of content and can be efficiently processed and fit well with existing technology. XML, used by the TAC system, is adequate as a low level language for encoding information but falls short as a language in which to express information at the knowledge level, even when augmented by more recent components such as XML Schema, XSL or through applications such as WSDL.

Our TAGA system uses OWL [Dean, 2002] as the content language for agent communication. Compared with RDF that used on our previous TAGA work [Zou, 2003], OWL has a well-defined model-theoretic semantics as well as an axiomatic specification that determines the intended interpretations of the language. OWL is unambiguously computer-interpretable, thus making it amenable to agent interoperability and automated reasoning techniques. The benefit of adopting a stronger semantically rich content language like OWL is that it facilitates a higher-level of interoperability between agents. By agreeing on how meaning is conveyed, it is simpler for applications to share meaningful content.

We have defined the OWL ontology for use as a FIPA-compliant content language. In addition to the basic required classes (e.g., Agent, ACLMessage, Service, etc.) and necessary expressive requirement (such as Proposition, Action, and Reification), our ontology provides supports for expressing rules, queries and responses to queries. We believe that OWL is a good choice as a general ACL content language for four reasons. First, its expressive power as a knowledge representation language seems to be adequate for many if not most needs of current agent based systems. Second, it offers better support for using terms drawn from multiple ontologies than do current popular ACL content languages. Third, as a semantic web language, it is designed to fit into and integrate with web-based information and service systems. Fourth, OWL has the potential to be a widely accepted and used representation language, enhancing the potential for interoperability among many systems. We will touch briefly on the first two points and leave the others as exercises for the reader.

To demonstrate that OWL is an adequate language for ACL content we consider a list of test cases presented in [Bothelo 2002]. These examples were used as an expressive test for a candidate FIPA content language and compared the result of encoding these in SL, KIF, ebXML, Prolog and DAML. Clearly OWL is less expressive than SL, KIF or Prolog, but the OWL version of these test cases given in Table 1 show that it's up to most of tasks it might be asked to serve.

Table 1: OWL Expressivity Test

Expression	**Representation**	**Comment**

"Schrödinger's Cat is alive"	<Cat rdf:ID="schrödinger-s_cat">
	  <owner>Shrodinger</owner>
	  <status> alive </status>
	</Cat>
		There is a live cat in the world whose owner is Shrodinger.

"Cats are animals"	<owl:Class rdf:ID="cat"> <rdfs:subClassOf rdf:resource="#animal"> </owl:Class>	Cat is subclass of animal
"You making the tea"	<fipaowl:Action rdf:ID="tea_action1"> <fipaowl:act>making-tea </fipaowl:act> <fipaowl:actor>you</fipaowl:actor> <fipaowl:Action>	There is a making-tea action, "you" are the actor.
"Drinking too much is bad for you"	<Behavior rdf:ID="drinktoomuch"> <hasBehavior>excessive_drinking</hasBehavior> <healthy>bad</healthy> </Behavior >	The behavior of drinking too much is bad for your health.
"All red things"	<owl:Class rdf:ID="allredthing "> <owl:intersectionOf rdf:parseType="Collection"> <owl:Classrdf:about="#Thing"/> <owl:Restriction> <owl:onProperty rdf:resource="#hasColor" /> <owl:hasValue rdf:resource="#Red" /> </owl:Restriction> </owl:intersectionOf> </owl:Class>	The things whose color are red.
"Any color a car might have"	<owl:Class rdf:ID="anycarcolor"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#color" /> <owl:allValuesFrom rdf:resource="#CarColor " /> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	The color that limits the color property value in the car colors. This can also be a query: "Select color where color in Car Color"
"All things are hot"	<owl:Class rdf:about= "#Thing"> <rdfs:subClassOf>	All things's temperature

	<owl:Restriction> <owl:onProperty rdf:resource="#tempterature"/> <owl:hasValue rdf:resource="#hot" /> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	are hot.
“Somethin g is cold”	<owl:Thing rdf:ID= “cold_thing”> <temperature>cold</temperature> </owl:Thing>	There exist something whose temperature is cold.
“Herring or Perch”	<owl:oneOf rdf:parseType="Collection"> <owl:Thing rdf:about="#Vokda "/> <owl:Thing rdf:about="#Perch"/> </owl:oneOf>	
“Vodka and Tonic”.	<owl:union Of rdf:parseType="Collection"> <owl:Class rdf:about="#Vodka " /> <owl:Class rdf:about="#Tonic" /> </owl:unionOf>	
“Not cricket”	<owl:Class rdf:ID="Noncricket"> <owl:complementOf rdf:resource="#Cricket " /> </owl:Class>	
“Success implies Payment”	<fipaowl:Rule> <fipaowl:Implies > <fipaowl:head > Payment</fipaowl:head > <fipaowl:body >Success </fipaowl:body > </fipaowl:Implies> </fipaowl:Rule>	The rule : Payment :- Success.
“Luis has the persistent goal that W”	<Person rdf:ID= “Luis”> <hasPersistentGoals> W </hasPersistentGoals> </Person>	
“Steve Believes X”	<Person rdf:ID=”steve”> < hasProposition> < Belief rdf:ID=”stevebelief1”> < believe>true</believe> < Statement > X</Statement> </Belief> </hasProposition> </Person >	
“Jonathan Desires Y”	<Person rdf:ID=”Jonathan”> <hasProposition> <Desire rdf:ID=”jonthandesire11”> <desire >true</desire> <Statement > Y </Statement> </Desire> </hasProposition> </Person >	
“Matthias Intends Z”	<Person rdf:ID=”Matthias”> < hasProposition> <Intend rdf:ID=”Matthiasintend1”> <intend>true</intend> <Statement > Z </Statement> </Intend> </hasProposition> </Person >	

Compared with other ACL content languages, OWL provides much better support in modeling, maintaining, and sharing ontologies. Standard content languages such as SL and KIF offer no explicit mechanisms for ontology support. FIPA inherited the simple mechanism for ontology specification first used in KQML that essentially required that all content terms in a particular message be tagged as coming from a single ontology. Although variations and “work arounds” to this constraint have been proposed, implemented and used, none have been formally adopted as part of the stable FIPA specification. OWL supports multiple namespaces and ontologies and, in fact, is a large part of its raison d'etre. Large scale and open multi-agent systems will benefit from OWL’s abilities to integrate information from different ontologies. Moreover, OWL and other semantic web languages, will better support other services essential to large scale open systems, such as the capability to translate or map information from one ontology to another and to negotiate meaning or otherwise resolve differences between ontologies.

3.3 Understanding Messages

When an agent receives an incoming ACL message, it computes the meaning of the message from the ACL semantics, the protocols in effect, the content language and the conversational context. The agent’s subsequent behavior, both internal (e.g., updating its knowledge base) and external (e.g., generating a response) depends on the correct interpretation of the message’s meaning. Thus, a sound and, if possible, complete understanding the *semantics* of the key communication components (ACL, protocol, ontologies, content language, context) is extremely important. In TAGA, the service providers are independent and autonomous entities, which makes enforcing a design decision that all use exactly the same ontology or protocol difficult, if not impossible. For example, the Delta Airline service agent may has its own view of travel business and uses class and property terms that extend an ontology used in the industry as a whole. This situation parallels that for the semantic web as a whole – some amount of diversity is inevitable and must be panned for lest our systems become impossibly brittle.

The ontologies in TAGA are distributed and managed by multiple parties. This distributed model is a better fit for deployment in an open web environment. There is no centralized site or agent that has to understand every ontologies. Ontologies and rules are designed and implemented by service owners to reflect their business models and meet their requirements; tan agent belonging to a service owner is responsible for answering the question related to the ontologies it uses. Ontologies store in local and may access only by local agent. We could define personalized ontologies and rules. It would help resolving the problem of security and trust.

Many of the agents we have implemented in the TAGA system use FOWL (Flora OWL) to represent and reason about content presented in RDF or OWL. FOWL is a flora-2 [Yang 2000] program that interprets RDF and OWL represented as a collection of RDF triples. Flora-2 is itself a compiler that compiles from a dialect of f-logic into XSB, taking advantage of the tabling, HiLog and well-founded semantics for negation features found in XSB. On receiving an ACL message with content in RDF or OWL, a TAGA agent parses the content into triples, which are then loaded into the XSB engine for processing.

The message's meaning (communicative act, protocol, content language, ontologies and context) all play a part in the interpretation. For example, receiving a query message using query protocol, the agent searches its knowledge base for matching answers and returns an appropriate inform message. TAGA uses multiple models to reflect the multiple namespace and ontologies in the system. The agent treats each ontology as an independent Model in XSB engine. The support of ontology sharing and exchanging is achieved by defining a set of ontology related actions:

- *NewInstance:* this message creates an instance using the specified ontology and the provided instance data;
- *OntologyQuery:* this message queries other agents about the terms defined in their ontology;
- *OntologyShare:* this inform message is about the ontology definition, which include Class/Property definition, Class-Subclass relation and Class-Property relation.
- *OntologyRelation:* this message is about the conversion and relations among class or property term defined different ontologies. For example, agent A informs agent B that the class Person is same class as the class Human used by agent B. The relations include extension, identical and equivalent. This message can be an inform message informing other agents about the relation, or query message asking to confirm the relation, or request message asking to translate the ontology term used in multiple ontologies.

3.4 Query Support

Among the most important communicative acts used by agents are those designed to support querying. The FIPA ACL has a very simple query model supporting just two acts -- *query-if* and *query-ref* – but allows a more complicated query to be encoded as a *request* act. In order to use semantic web languages for ACL content, we have experimented with the integration of a number of RDF based approaches, including DQL, RQL, RDQL, Triple, and TAP. Since a consensus query system has not yet emerged, we have adopted an approach in which agents can use any of several query systems and associated protocols. An agent specifies the query languages and protocols it understands as part of its basic service description. Other agents who intend to submit query to this agent are expected to encode the query string in one of the support languages. Table 2 is a query and answer example using RDQL language.

Table 2: Query and answer example

Query:

```
<fipaowl:Query  rdf:ID="query1">
        <fipaowl:queryLanguage>rdql</fipaowl:queryLanguage>
          <fipaowl:question>
                    "SELECT ?x,?y
                    FROM <people.rdf>
                    WHERE (?x,<dt:friend>,?y),(?y,<dt:friend>,?x)
                    AND ?x<^gt;?y
                    USING dt for <http://foo.org#>, rdf for
                    http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          </fipaowl:question>
          <fipaowl:result_number>10</fipaowl:result_number>
 </fipaowl:Query >
```

Answer:

```
<fipaowl:Query  rdf:about="query1">
   <fipaowl:queryLanguage>rdql</fipaowl:queryLanguage>
          <fipaowl:result_number>1</fipaowl:result_number>
          <fipaowl:answer>
                    "Array ( [0] => Array ( [?x] =>
                    http://foo.org/persons/Carl [?y] =>
                    http://foo.org/persons/Peter ) [1] => Array
                    ( [?x] => http://foo.org/persons/Peter [?y]
                    => http://foo.org/persons/Carl ) )"
          </fipaowl:answer>
 </fipaowl:Query>
```

We have found that the basic framework of FIPA standards support this approach well by having a good set of primitive communicative acts, a way for agents to define communication protocols, and a sound mechanism by which agents can describe their capabilities and the supporting services. We are planning to experiment with adding mediator agents to TAGA that offer a query translation service. Such an agent would be able to handle several kinds of query languages permitting it to act as a proxy. For example, agent A might wish to ask a DQL query of agent B, which only understands RQL. A query translation service able to process both DQL and RQL could provide the mediation service – receiving a DQL query from A, sending appropriate RQL queries to B, accepting the response, and reformulating to fit the DQL protocol.

4. DISCUSSION

In this section we will briefly discuss several additional design issues we have addressed in TAGA.

Ontologies. In addition to the FIPA content language ontology, we have defined two domain ontologies in OWL. The first is a travel ontology that covers the basic concepts of traveling needed in TAGA, include the travel itinerary, customers, travel services and service reservations. The second ontology is one for auctions. This ontology is used to define the different kinds of auctions, the roles the participants play in them, and the protocols used.

Service description and matching. FIPA agents are associated with one or more FIPA platforms, each of which offers a set of agent services including a Directory Facility (DF) agent that handles service registration, deregistration and matching. When an agent registers a service in a DF, it provides service information like the service type and owner. However, more specific service information may also be useful when searching for agent services. For example, a customer may want a booking in a hotel with at least three star rating, is close to public transportation, offers breakfast, and accepts VISA card payments. This can be achieved with the use of DAML-S [DAML-S, 2002] profile. In TAGA, every travel service provider describes its service process model with DAML-S language and publishes it as a web page. This covers basic service information like address, phone number and service interface information. For example, a hotel may describe booking service as: customer name, payment methods, travel date as input; reserve number as output; the effect of booking is one room occupied at the travel date. The travel agent, who is responsible for organizing travel package, is able to contact with customer agent and related service agents and finds the best match. First the travel agent loads the DAML-S parsing rule and planning rules into its XSB reasoning engine. It then loads service agents' DAML-S profiles and customer's personal profile. The best matching service providers are selected and a most profitable travel package is composed dynamically.

Implementation comments. The original Trading Agent Competitions relied on a few centralized market servers to handle all interactions and coordination, including service discovery, agent communication, coordination, and game control. In contrast, the TAGA framework uses a distributed peer-to-peer approach based on standard agent languages, protocols and infrastructure components (FIPA, Agentcities), emerging standards for representing ontologies, knowledge and services (RDF, OWL, DAML-S) and web infrastructure (e.g., Sun's Java Web Start). Several FIPA platform implementations are currently used within TAGA, including Jade and AAP (April Agent Platform), demonstrating agent interoperability. Our current demonstration system allows new users to dynamically join a running game at any time. A dummy agent implemented in JADE can be downloaded and run to instantiate a new TA agent. A simple GUI allows the user to set operating parameters or the java code can be modified or extended. A set of web based monitoring services allow one to see the status of a game, examine messages being sent, lookup the reputation of agents, etc.

Contribution. We see two main contributions in our work. First, TAGA provides a rich framework for exploring agent-based approaches to e-commerce like applications. Our current framework allows users to create their own agent (perhaps based on our initial prototype) to represent a TA or SA and to include it in a running game where it will compete with other system provided and user defined agents. We hope that this might be a useful teaching and learning tool, not only for multi-agent systems technology, but also for the semantic web languages RDF and OWL and their use in agent based systems. Secondly, we hope that TAGA will be seen as a flexible, interesting and rich environment for simulating agent-based trading in dynamic markets. Agents can be instantiated to represent customers, aggregators, wholesalers, and service provides all of which can make decisions about price and purchase strategies based on complex strategies and market conditions. Moreover, simulations like TAGA encourage exploring aspects of e-commerce that go beyond auction theory. TA agents might compete on their ability to better understand the descriptions of services sought and services offered and the basic models of the preferences of their users in order to best satisfy the needs of their clients. These descriptions, of course, will be in a semantic web language like OWL.

5. Conclusions and future work

Travel Agent Game in Agentcities (TAGA) is a framework that extends and enhances the Trading Agent Competition (TAC) system to work in Agentcities, an open multiagent systems environment of FIPA compliant systems. We hope that TAGA will serve as an experimental test-bed for several communities of users. First, it provides an environment, which can be used to explore aspects of multiagent systems technology based on the mature, published FIPA standards. Research on multiagent systems technology is best done with in a rich yet easily understood problem domain. We have found that the travel agent scenario as originally put forth by TAC provides both the richness as well as accessibility, especially when opened up to be peer-to-peer. We are using TAGA as a test-bed for research on the use of semantic web languages (e.g., RDF and OWL) as content languages and as service description languages. Future work is planned in adding more sophisticated negotiation and ontology mapping to our TAGA environment.

Second, we hope that TAGA could serve as an interesting framework and test-bed for experiments with automated markets and trading. By adding autonomous service provide agents (e.g., for hotels) one could experiment with a dynamic market with both "shopbots" and "pricebots" or investigate the role of intermediation in the form of agents performing a wholesale function.

Third, we hope that others will find TAGA useful as a test, demonstration and teaching environment, both in technology classes focused multi-agent systems, FIPA standards or the semantic web and in business or e-commerce classes focused on automating commerce and trading, auctions or agent-based simulations.

The Agentcities project is exploring the delivery and use of agent-based services in an open, dynamic and international setting. We are working to increase the integration of TAGA and emerging Agentcities components and infrastructure and will include agents running on handheld devices using LEAP.

6. REFERENCES

[Bothelo 2002] L. Bothelo, S. Willmott, T. Zhang, J. Dale, A review of Content Languages Suitable for Agent-Agent Communication, EPFL I&C Technical Report #200233.

[Dale, 2002] J. Dale, S. Willmot, and B.Burg: Agentcities: Challenges and Deployment of Next-Generation Service Environments. Proc. Pacific Rim Intelligent Multi-Agent Systems, Tokyo, Japan, August 2002.

[DAML-S, 2002] The DAML Services Coalition (alphabetically Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara): DAML-S: Web Service Description for the Semantic Web, The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.

[Dean, 2002] M. Dean and Guus Schreiber (eds): OWL Web Ontology Language 1.0 Reference. W3C Working Draft.

[Greenwald, 2001] Amy Greenwald and Peter Stone: Autonomous Bidding Agents in the Trading Agent Competition, IEEE Internet Computing, March/April 2001.

[Greenwald, 2003] Amy Greenwald (ed.). The 2002 trading agent competition: An overview of agent strategies. AI Magazine, to appear

[Sagonas, 1994] Kostantinos Sagonas, Terrance Swift, and David S. Warren: XSB as an efficient deductive database engine, In ACM Conference on Management of Data (SIGMOD), 1994.

[Stone, 2000] Peter Stone and Amy Greenwald: The First International Trading Agent Competition: Autonomous Bidding Agents, Electronic Commerce Research Journal pp1-36, 2000.

[Wellman, 2002] Michael P. Wellman, Amy Greenwald, Peter Stone, and Peter R. Wurman: The 2001 Trading Agent Competition, Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI-2002), pp935-941, Edmonton, August 2002.

[Willmott, 2001] Willmott, S., Dale, J., Burg, B., Charlton, P. and O'Brien, P., Agentcities: A Worldwide Open Agent Network. In: AgentLink News, Issue 8, November 2001.

[Yang, 2000] Guizhen Yang and Michael Kifer. FLORA: Implementing an efficient DOOD system using a tabling logic engine. Proceedings of Computational Logic --- CL-2000, number 1861 in LNAI, pp 1078--1093. Springer, July 2000.

[Zou, 2003] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan, TAGA: Trading Agent Competition in Agentcities, Workshop on Trading Agent Design and Analysis, held in conjunction with the Eighteenth International Joint Conference on Artificial Intelligence, Monday, 11 August, 2003, Acuulco MX.