An Implementation of Auctions in 3APL

Norman Salazar Ramirez
Benjamin Auffarth
Sergio Alvarez Napagao
Universitat de Barcelona
course report for *Multi-Agent Systems*
at Universitat Politècnica de Catalunya

January 26, 2007

Abstract

We implemented two auction frameworks in 3-APL in order to test its capabilities. We provide some criticism, including concerning the IDE, the parser, the lack of more sophisticated data structures. We conclude that experience has been quite good, and we think that, as it is noted before, 3-APL could be a great language for BDI agents if this kind of problems were solved.

Contents

1 Introduction

3-APL stands for *Abstract Agent Programming Language* or *Artificial Autonomous Agents Programming Language* and was created in the University of Utrecht as an academic tool for developing intelligent agents. The intelligence of the agents is provided by a complex mental state, which consists of different mental attitudes, and by a deliberation process that provides a transition mechanism between the mental states. Also the agents are capable of interacting with each other, directly by communication, or indirectly through the shared environment. The platform part of 3-APL provides the mechanism to deploy multiple agents at the same time as well as managing their communication.

The purpose of this project was to find out about the worth of 3-AOL as a multi-agent language and developing tool. Our reason for asking this question, come from the fact, that most of the literature we found about 3-APL, only covers its theorical properties, its plans for the future, and its design; and no mention or reference to examples or implementations is given, exempt of some toy code that is already provided with the tool. (e.g. (Dastani, Riemsdijk, & Meyer, 2005), (Dastani, Boer, Dignum, & Meyer, 2003), (dInverno, Hindriks, & Luck, 2000))

At first glance we found the language attractive, with respect to its reasoning operators and capabilities; however a second glance gave us the impression that as a language and tool, 3-APL was lacking in important aspects related to communication, procedural methods, integration with its underlining java and it friendliness. So we deem necessary taking a third glance to what 3-APL represents as a multi-agent developing tool.

This study will consist in implementing a simple, yet interesting (non-trivial) multi-agent system, so that a good grasp of its capabilities and short-comings can be accomplished. We choose as a system to model a market place/auction; since it is a well known problem in the literature that has been treated by multi-disciplinary approaches, and has many of the properties and requirements which are useful in multi-agent systems (e.g. reasoning, communication, negotiation/cooperation, among others).

In the following sections a small overview of what an auction is, and of its different types; followed by a recapitulation of what 3-ALP is. We will describe exactly what kind of auctions we tried to model and the methodology we applied to create this model from an agent point of view; finally we will talk about how the actual implementation proceeded, present some discussion topics and conclusions.

2 Methodology

The system that we have designed and implemented is a small auction market in where agents can offer and bid for products. We have developed two basic prototypes, which are described in this section. In auctions, participants bid openly against one another, with each bid being higher than the previous bid. The auction ends when no participant is willing to bid further. Auctions have been studied in a multi-agent framework, e.g. (Sandholm, 1995), (P. R. Wurman, Wellman, & Walsh, 1998), and (P. Wurman, Walsh, & Wellman, 1998)

2.1 Simple buyer – seller

In our first protoype, we have designed a simple system in which there are two types of actors:

- Sellers: proactive agent that has a list of objects to sell, which are offered once at a time to a list of possible buyers.
- Buyers: reactive agents, each buyer has a happiness threshold, a list of the utilities each object has for itself, and a matrix of utilities each object it believes the other agents to have.

The initial procedure of as follows:

- The seller checks if it has objects to be sold. If there are any, it selects one. It stops otherwise.
- The seller sends a broadcast message to everybody in its buyer list, offering the object.
- Each buyer, as it receives the offer, checks the utility of the object being offered and the money it has available.
- If there is sufficient money and if the utility high enough, it makes a bid.
- The seller receives bids and notifies bidding agents if they are not the highest anymore.
- Some buyers may refrain from further bidding and pass.
- After some time, the seller accepts the highest bid.
- The object is moved from seller to buyer in exchange for the pay that comes from contrary direction. Both buyer and seller increase their happiness corresponding in extend to utility and money gained, respectively.
- If Buyers' happiness exceeds a threshold they quit the auction (they are already happy enough).
- The seller goes on selling the next object.

2.2 Host – Seller – Buyer

We tried to make the problem a little more interesting and complex, by adding a new type of participant to the system, the auction host. The role (not to be confused with the concept of role in MAS) of the host, is that if regulating and directing the participants in the auction, or in more simple terms, it adds a level of bureaucracy to the system. For our testing purposes this bureaucracy lets us play with a more complex communication scheme, which is often criticized in 3-APL and represents one of our testing objectives.

In this schema the seller and buyer players retain the same characteristics and objectives as before, however new requirements and restrictions are placed upon them; these restrictions will be placed by the new player, the host. Now whenever a new buyer player enters/exits the system, it is required of him to present himself to the host; otherwise it will not be able to participate in any auction.

A similar scenario is required of the sellers. Whenever a seller wants to offer a good for sale, it will need to notify it to the host, who will then put it on a queue list of auctions to do. The host then selects a seller from its list of sellers, (this can be done by some simple heuristic, or to optimize certain function) and announces to all the buyers in the system, that an auction for item X will start and that if interested the seller S should be contacted. After this, the process of the auction remains the same as explained before, and its only when the item is sold (or certain time passes, if no biddings where done) that the control returns to the agent. This can be complicate even more by allowing the host to start multiple agents at the same time and by allowing buyers to bid in the multiple auctions.

In difference with the other two type of players is that the host does not try to maximize a function (happiness), but connects the agents and controls most of the process.

3 Design

We used Prometheus as the multiagent design tool. The Prometheus tool is very easy to use and powerful which rendered this process very comfortable for us. As it was our first experience with it, some features of it may have not been used at its best, but it was very useful for our later implementation purposes.

The full design for our system is provided as HTML together with this work and additionally available at `http://www.lsi.upc.es/~salvarez/auctions/`.

4 Implementation

3-APL is not only the language but the platform that allows programming, deployment and execution of the agents. The 3-APL platform is also in charge of the transportation of the communication messages; and also provides information about the existing agents to other agents through the agent management system

(AMS). Moreover, once the agents are running it provides monitoring tools, such as a sniffer, for the message exchange and specific windows to monitor the mental states of the agents. It also provides means for step by step monitoring.[1]

On the specification side, 3-APL requires Java 1.5 to run, and it is it is easy to install by downloading and executing an 850kb.jar file. During deployment time it can be set as it can be set as a server to host multiple agents or as a client.

We found some strong points in 3-APL that make it look a strong, versatile and promising language if it is to be continued and improved as a project. The most remarkable points are a robust and extremely simple communication platform, a very intuitive and powerful language, and a whole new world to be explored with the Java and Prolog bindings. This last point is very important for us. Prolog supports the creation of shared, extensible and complex belief bases. Java is a key advantage, as it allows 3-APL to interact with any other Java software, opening a window to resources and devices.

There are some major issues in 3-APL, though, that present grave disadvantages at the implementation stage. The main concerns are related to the programming tool bundled with the release and which is the only way at the moment to build and compile 3-APL projects. For instance, the tool does not have copy and paste capabilities, and some characters were not mapped to our keyboards, even the Tab key did not work. So we had to work with external editors at all times, making the implementation process much slower.

Furthermore, the integrated compiler, often at a time, did not show proper parser messages, so we wasted many hours trying to find the exact parenthesis or colon we were missing in the code. As a last example, there is no support for more sophisticated data structures, even lists are uncomfortable to use and render the code into spaghetti of routines that are only concerned in reading them. 3-APL lacked also nested lists, which forced us to implement our own routines to use them. Communication is simple and powerful by passing a belief to the receiver's belief base, but it cannot be bound to an ontology and does not use performatives. This could make it weak inside multiagent systems which require a strong use of semantics.

However, the overall experience has been quite good, and we honestly think that, as it is noted before, 3-APL could be a great language for BDI agents if this kind of problems were solved. We could solve some of them, as in the previously said example of the nested lists: the capabilities and rules system can be a bypass to code procedural methods, which is a good point.

[1]A more detailed description can be found at `http://www.cs.uu.nl/3apl/download/java/userguide.pdf` and `http://www.cs.uu.nl/docs/vakken/map/slides/3apl-Syntax.pdf`

5 Conclusions

We think it became apparent that 3-APL is a powerful tool for the creation of intelligent agents, especially its practical reasoning rules provide ways of creating very complex mechanisms. However the use of static deliberation cycles[2] limits its functionality and forces the user to explode the number of rules to balance this out, which complicates the task of agent implementation.

The lack of a strong communication protocol also is a further point which lacks in the language, complicating the task of creating coordination and cooperation between agents.

We also find lacking the absence of agent actions dedicated to computer processing; right now if an agent requires an action not related with changing its mental state it has to be done through the environment. For example it would be nice to have agents in which part of its executing plan will consist of running some classification algorithm or some numerical method which is only owned by the agent and so it could be encapsulated in its program.

3-APL is a comparatively new model (first publications date back to 1998). It has been noted that there are no industrial-strength applications yet, which could boost its development.

References

Dastani, M., Boer, F. de, Dignum, F., & Meyer, J. (2003). Programming agent deliberation: An approach illustrated using the 3apl language. *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), Melbourne*.

Dastani, M., Riemsdijk, M., & Meyer, J.-J. (2005). Programming multi-agents systems in 3apl. *Multi-Agent Programming (Languages, Platforms and Applications)*, 39–67.

dInverno, M., Hindriks, K. V., & Luck, M. (2000). A formal architecture for the 3apl agent programming language. *Proc. of ZB00*, 168-187.

Sandholm, T. W. (1995). Limitations of the Vickrey auction in computational multiagent systems. *Proceedings of the First International Conference on Multi–Agent Systems, MIT Press*.

Wurman, P., Walsh, W., & Wellman, M. (1998). *Flexible double auctions for electronic commerce: Theory and implementation.*

Wurman, P. R., Wellman, M. P., & Walsh, W. E. (1998, 9–13,). The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In K. P. Sycara & M. Wooldridge (Eds.), *Proceedings of the 2nd international conference on autonomous agents (agents'98)* (pp. 301–308). New York: ACM Press.

[2]See `http://www.cs.uu.nl/3apl/deliberationcycle.pdf`.