A PVM implementation of the Fishmarket Multiagent System

C. Di Napoli M. Giordano M. Mango Furnari
Istituto di Cibernetica - Naples (Italy)
C. Sierra, P. Noriega
IIIA - Barcelona (Spain)

Abstract

This is an implementation proposal for a Structured Negotiation Environment for Heterogeneous Agents (SNE-H) based on the Fishmarket metaphor. Our aim in this implementation is to explore design issues that may be relevant for the construction of a test-bed for heterogeneous agent architectures. Hence, the Fishmarket multiagent system has been built on PVM as a prototype for a family of even more sophisticated multiagent interaction environments.

1 Introduction

In this paper we describe our implementation of a Structured Negotiation Environment for Heterogeneous Agents (SNE-H) based on the Fishmarket metaphor. The SNE-H [6,8] is an electronic market-place where certain kinds of goods are traded under explicit trading conventions. This environment is structured in the sense that the trading conventions impose explicit objective requirements to the admission of goods and traders into the market-place, and restrict the agents' behaviour to only those actions that are necessary and sufficient for actual trading under those conditions.

In the Fishmarket metaphor there are many participants, each one performing a particular job. We identify, at the moment, these agents: Buyers, Sellers, Auctioneer, Buyer Admitter, Seller Admitter, Closing Agent, Supervisor.

The Sellers are ordered according to the time at which they actually enter the market. Each Seller submits his goods to the referring Admitter, who decides the goods quality, weight and price. Then the Seller does not take part in the market actions until his goods have been sold. Only when this happens, he receives the money from the Closing Agent.

In order to participate in the auction, a Buyer must have an auction registration. So he first contacts his referring Admitter and asks for a registration. He also goes to the Closing Agent in order to open an account for auction payment.

The goods list, scheduled by the Seller Admitter, is passed to the Auctioneer, who is able to change the goods parameters settled before. Then he can start the auction from the next goods to sell. At this point, the most fascinating market action takes place: the "Bidding Round". The Buyers look at the Auctioneer who decreases the goods price of the current Bidding Round very speedly until some Buyers rise their hands. If there is only one Buyer bid then he is the winner, while if there are more conflicting bids, then the Auctioneer restarts the Bidding Round with the current price increased by a 25%. There is also another possibility: the Buyer has not enough money to pay. If so, the Bidding Round can restart from the initial price, while the Buyer may pay for his fault. While the bid conflicting situation is managed by the Auctioneer, the latter one is detected by the Supervisor.

Only when there is a "real" winner, the goods are awarded to him and other Bidding Rounds can take place. After the bidding is complete, the Buyer goes to the Closing Agent to collect his goods while the latter updates his account.

The paper is so organised: in section 2 we describe a simplified version of the Fishmarket we used in our implementation; in section 3 we discuss the reasons why we used the PVM software package to perform all the inter-agents communication in a distributed setting; in section 4 the actual implementation is described; finally some possible system improvements are reported.

2 The Fishmarket protocol design

The Fishmarket Multiagent protocol implementation we worked on is based on the PVM [1] (Parallel Virtual Machine) package. We designed the implementation starting from a simpler agent classification, where each agent has the following jobs:
In this simplification the same Admitter deals with the admission of both Sellers and Buyers admission, and it also accomplishes the Closing Agent and Supervisor work. The Auctioneer work is unchanged.

This classification, although easier than the original one, does not affect the protocol complexity in a significant way. In fact, this simplified multiagent system organization captures most of the agent functionalities and communication aspects which are characteristic of the Fishmarket interlocution protocol.

As shown in figure 1, the multiagent interaction in the fishmarket protocol is a temporal sequence of scenes, where each scene is characterized by some participating agents and a well-defined interlocution protocol. We single out three main scenes:

<table>
<thead>
<tr>
<th>Admission Scene</th>
<th>Auction Scene</th>
<th>Closing Scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>participants: B, Ad</td>
<td>participants: B, Ad, Au</td>
<td>participants: B, Ad</td>
</tr>
<tr>
<td>actions: 1) Ad sets the auction goods. 2) B submits the auction registration (or removes it) to Ad; 3) Ad updates the set BA of Buyers registered into the auction; 4) B asks Ad to update its credit;</td>
<td>actions: 1) Au gets the next goods to sell from Ad; 2) Au gets the BA list from Ad; 3) Au sets the price and starts the Bidding Round (init auction); 4) Au receives bids from BA (&quot;lock&quot;); 5) If &quot;bids collision&quot; then goto 2; 6) Au asks Ad to verify the purchase; 7) If &quot;bad purchase&quot; then goto 8 otherwise goto 9; 8) Ad gets off the Buyer from BA - goto 2; 9) Au notifies Ad the bidding is complete; 10) Au closes the auction.</td>
<td>actions: 1) B gets the goods from Ad. 2) Ad updates the credit of B after the sale;</td>
</tr>
</tbody>
</table>

In the protocol design the Auction Scene is the core of the agent negotiation, while the other scenes open and close each Bidding Round.

The Auction Scene is a sequence of Bidding Rounds. Each round is started by the Auctioneer only after he has settled the price of the goods (init auction). A Buyer can change scene at any moment, but there are some constraints, as we show in figure 1. A Buyer cannot enter the Auction Scene without performing the registration procedure in the Admission Scene. Moreover he cannot enter the Closing Scene
without being a "Bidding Round Winner". However, he can leave the Bidding Round whenever he wants, entering, for instance, the Admission Scene in order to update his account.

We decided to implement this multiagent protocol on a Network of Workstations. On top of this support we used the Parallel Virtual Machine (PVM) package to provide a programming environment for cooperative programs running on different computer systems.

3 Why we used PVM

The Fishmarket Multiagent protocol is characterised by:

- **dynamicty** - while the presence of the Admitter and the Auctioneer is necessary for the market’s existence, the Buyer set (the registered Buyers) can change dynamically during the market session.
- **autonomy** - while the Auctioneer and the Admitter are some sort of "automata" as they react automatically to events (except for some cases in which they can take decisions), the Buyers are autonomous agents, who can freely decide, at any moment, to leave or enter the auction session, to buy or not the goods, to update their accounts, and so on. The protocol implementation should guarantee this freedom.
- **non-determinism** - Buyers have a “human” behaviour, since they act according to some “strategies” and “reasoning” but also to their “feelings” and “intuition”. This makes impossible to predict the system evolution.
- **asynchronous mode** - During the Fishmarket session, Buyers can ignore or react to outside events. This freedom affects the communication mode which is essentially asynchronous.
- **interaction** - the Bidding Round is the situation with the most communication traffic. We can imagine how chaotic a hall should be crowded with many people, looking at a single man who screams the next prices, and all Buyers are ready to rise their hands, and often the man has to decide among different bids. In the other scenes the interaction is easy as it involves two agents, and usually the interlocution is characterised by a request/answer protocol.

We think that PVM [1] can afford most of the features we have reported. In fact, the PVM parallel programming environment is characterised by a high dynamicity degree for the configuration of both the virtual machine and the running task pool. PVM provides routines to add and delete new machines during the computation, to manage the unexpected virtual machine changes (due, for instance, to a computer crash), to spawn and kill processes.

According to the autonomous agent feature, any process, started outside a PVM session, can enter it by contacting the local PVM daemon. This daemon has a local task table which keeps information about all local processes. This table can be consulted from inside any program. This means that any program, running autonomously on a machine, can contact any other program registered into the PVM session and then create a communication channel with it.

The PVM paradigm allows both synchronous and asynchronous communication modes. A process can either block its computation until a message arrives on the receiving buffer or repeatedly check the buffer during its computation. It may also decide to synchronise itself with the others on the message arrival, i.e., each process stops its computation until all the others have received the message (synchronisation barrier).

With PVM it is very easy to create new communication channels, allowing more control over the message traffic. Moreover there is no need for a program to cope with all incoming messages. If needed, they can be ignored or destroyed using the PVM sending and receiving buffer routines.

PVM allows both one-by-one and broadcast (multicast to groups) communication modes. This is useful respectively when you deal with the interaction among few processes or very intensive communication patterns.

The advantages of using the PVM package are several. First, it allows heterogeneous computer systems connected through a network to be viewed as a single parallel virtual machine. It is very easy to build such hardware support without worrying about compatibility issues among different computer systems.

PVM has a sophisticated control system to guarantee fault tolerance, to manage machine crashes, unexpected configuration changes, to control message routing, synchronisation, etc. The transmission reliability is guaranteed by the low layer TCP/IP protocol, together with the network security features.

Through the use of PVM the market can be distributed over a LAN, making agents cooperate over the world, in a physically distributed market. For applications with less intensive communication, and mainly characterised by cooperation, a computational platform distributed over the world could be attractive. Nevertheless, the possibility of reconfiguring dynamically this computational support may be a great advantage for these applications.
4 General description of the implementation

In our implementation, agents are C programs linked to the PVM libraries. As mentioned in section 2, we identified three kinds of agent: Admitter (Ad), Auctioneer (Au), and Buyers (B). For each agent we wrote a C program performing the specific agent computation as well as the communication by means of a set of routines which use the PVM communication primitives. These routines transparently handle the agent program's entry (and exit) into the community (dynamically reconfigurable), the message routing and data exchange (registration and negotiation messages), the errors and system faults management, and so on. The PVM code is entirely embedded in these communication routines that can be easily reimplemented using other software packages for distributed computing like P4 [2], Express [3], MPI [4].

Our aim is not the definition of standard communication primitives for multiagent systems. This is a great effort which should take into account many aspects like portability to different agent computational paradigms (KDB, Prolog, Lisp systems, etc.), general communication protocols (not only the negotiation one), exhaustive communication primitive set (to allow implementation of any kind of multiagent metaphor), expressiveness, and so on. A related work is KQML [5] (Knowledge Query and Manipulation Language) which is a language and a protocol to support interoperability among intelligent agents in distributed computing settings.

The purpose of this work is to explore design issues that may be relevant for the construction of a test-bed for multiagent structured negotiation environment. The Fishmarket metaphor is an example of heterogeneous and complex multiagent system which can be studied and implemented on parallel distributed memory architectures. Moreover we think that this work can be considered as a first simple prototype for even more sophisticated multiagent interaction environments in distributed settings.

4.1 Fishmarket abstraction and main components

![Figure 2: The Fishmarket Representation](image)

The target hardware architecture is a cluster of Workstations (Sun Sparcstation 5-10-10 514MP-20) interconnected through a network (Ethernet). When the user starts the PVM session, a PVM daemon will run on each machine in the cluster. Each daemon represent (see figure 2) a software interface for programs cooperating and running on different machines.

In our abstraction (see figure 2), the Fishmarket is composed by a "building" (the PVM daemon community) which is the "physical space" where the market takes place, and by the "employees", i.e. the Ad and the Au agents (programs running inside the PVM session).

In order to start the market we first need to start the PVM session, by starting the PVM console. This causes the activation of one daemon on the first host. If we want to "enlarge" the market to other network "sites" we need to add more hosts from the PVM console. In this way we set the starting PVM daemon community.

At this point, the market has a "physical" and "distributed" space. Then we need to make the "market employees" working, that means "spawning" from the PVM console the Ad and Au agent programs (which may run on two different machines). So the market is ready to run.

B programs are started outside the PVM session and each one may run on a different machine of the cluster. The user may start at any moment the B program and then decide to enter (or leave) the market.
Before going into details about the system behaviour and each agent program functionality, we have to underline some important differences among agents’ implementation. The way the Ad and Au programs behave in the Fishmarket session, is completely automatic. The Ad program loops forever waiting for incoming requests and reply to these requests with some actions as (Buyer registration, next sale to be sent to the Auctioneer, Buyer bid verification, etc.). In our implementation this program communicates with the other agents without any interaction with the user.

Also the Au has an automatic behaviour. This agent computes only during the Auction Scene, and most of its operations obey a well-defined auction protocol. So it automatically reacts to Buyers bids, resolving conflicts, asking the Ad to verify the bid, restarting the Bidding Round when the price decreases to zero, and so on. When the auction stops, the Au becomes idle, until the user starts a new auction. In order to allow this interaction between the user and the Auctioneer program, we developed a graphical interface (using the TCL/TK [7] package) for this program. So the Au program displays in a window the auction-history information and two buttons for the auction start and stop operations.

While the former agent programs are essentially automatons, the Buyer one is more like a software (and graphical) interface to support the "human buyer" (the user) interaction with the market distributed over the network. The only operations the B program performs deal with communication (registration requests, credit updates, buy goods, etc.). There is not a real computation. The Buyer program is like a complex graphical interface with many windows and buttons linked to subroutines performing the actual communication and data exchange. The agent executes in a non-deterministic in the sense of the user is free to decide at any time the action to do, and each action has a different effect on the global system.

4.2 System behaviour

To give a complete description of the system functionality we reported some pictures of what is on the display when the system runs. We ordered them according to the sequence in time of the market scenes.

![Figure 3: The Admission Scene](image)

In figure 3 we show the graphical interface of the B program when it is started by the user. By clicking the "Enter Market" button the B program looks for the market "existence", i.e. it looks for a PVM daemon running on its machine and checks if the Ad and Au programs run somewhere in the network.
If this is the actual situation, the B program enters the PVM session and it takes the identifiers of the Ad and Au programs. The “Info” menu of the B program main window gives information about the market agents and the auction history. PVM uses integers called task identifiers and host identifiers to uniquely identify tasks and hosts within the virtual machine. One entry (named “Market”) of the “Info” menu shows, for the B, Ad and Au programs, the name and the identifier of the host where each program is running, and the identifier of the program itself.

If the market does not "exist", the B program is not allowed to enter the market and it keeps on running in a non-cooperative mode.

Once the B has entered the market, it is ready to enter the Admission Scene. In the more complex Fishmarket design there should be a previous scene representing the Sellers’ admission to the market, during which Sellers enter the market and submit their goods to the Seller Admitter agent who sets the goods quality and price. In the actual implementation this scene has been ignored and the Ad program sets a default list of goods.

During the Admission Scene each B program asks for a registration in order to participate in the auction. As shown in figure 3, the B program first sets its market account by using the “Account Setup” entry of the “Actions” menu. The market account is a couple of numbers: the first is the Buyer “Credit” (i.e. the amount of money for goods purchases); the second is an “Aptitude” parameter (i.e. the Buyer reputation).

Then the B program may request a market registration using the “Admission” entry of the “Action” menu. This is the first action involving a communication between the B program and another agent (the Ad agent). Clicking the “Ask Registration” button, the B program sends the Ad a registration request, its credit, and its identifier. The Ad program receives them and puts all this information into the registered Buyer list (i.e. the auction current participants). Finally, the Ad sends the Buyer the registration notification.

If the Buyer performs a registration action without a previous account setting, it will be admitted to auctions with "zero" credits. This means that it will not be able to buy anything.

![Figure 4: The Bidding Round](image)

After the Admission Scene each B program is ready to enter the Auction Scene (see figure 4). When the Au program starts a new auction, for each registered Buyer an “Auction Window” will appear displaying the sale information, like “Seller Name”, “Seller Number”, “Goods” type, and “Last Price”. On
this window the price of the goods decreases very quickly. When the user decides to buy the goods at the current price, he pushes the "Buy" button and this causes the disappearance of his "Auction Window" and of all the other windows referring to the registered Buyers. If two or more Buyers pushed the "Buy" button at the same time, there is a conflict and the window will reappear representing the Bidding Round repetition with a starting price equal to the conflicting price increased by 25%. If nobody will buy the goods, the window disappears when the price becomes zero. Also in this situation the Bidding Round is repeated but with the original starting price.

The "life-time" of the Auction Window is the Bidding Round time. Once involved in a Bidding Round, a Buyer can do nothing else. We say that the B program is "locked". During this round, the Auctioneer sends the Buyers the goods price changes in a very fast way. This is one of the main characteristic of the Fishmarket protocol. In order to achieve this speed in the implementation we made the choice to "lock" the Buyers: before each new Bidding Round starts, the Auctioneer requests the Admitter the list of current registered Buyers, which are those the Auctioneer must communicate to; since the Auctioneer does not request any Buyer list update during the Bidding Round, it assumes that all the listed Buyers will participate in the Bidding Round until its end. So a Buyer is not allowed to update registration during this round. Though this choice introduces some constraints on the Buyer freedom during this scene, it allowed us to cope with the time issue, which is crucial in the Fishmarket example.

![Figure 5: The Auctioneer Agent](image)

In figure 5 we show the Au program graphical interface, with all the information concerning the auctions' history. As we told before, the Au program is driven by the user only for starting and stopping each Bidding Round. The stop event can be determined either by the user (who drives the Au graphical interface) or at least by one Buyer bid. In the first situation a dialog window will ask the user about the
auction repetition while in the second one the auction will restart automatically. Also in case of a "zero" price condition the auction repetition is performed in a dialog mode. A new auction cannot start when one of these situation happens:

- there are no more goods to sell,
- there are too few registered Buyers (the minimum number is 2),
- the starting price is too low (the minimum price is determined by the system speed).

When a Bidding Round ends, a registered B program is "unlocked" until a new one starts. During this time it may ask the Ad for a registration deletion or a credit updating. Of course, once the registration deletion has been performed, there will not be an "Auction Window" for the next auction.

When an auction is stopped by a Buyer bid, the Au asks the Ad program to verify it. The Ad program receives the bid information (Buyer identifier and price) and verifies if the Buyer credit is enough for that purchase. If it is not, the Ad puts the Buyer (the "cheater") out of the auction by deleting its registration and then the Au restarts the auction. Otherwise, in case of right purchase, the Ad updates the Buyer credit and the Au awards the goods.

4.3 Some considerations

In the Fishmarket system Buyer programs are started by users outside the PVM session and initially they have not communication patterns with other processes. Then, Buyer programs contact other agents for cooperation (which means to know "where" to send their messages). The way they identify the interlocutor, is to get process identifiers from the task table handled by the PVM daemon. Usually, in the PVM programming philosophy, the communication patterns are established among processes either via an activation chain (among parent and child processes) or via the grouping operation (by using the PVM grouping facilities). So the way the Buyer programs contact the Admiter and Auctioneer agents is not usual in the PVM programming philosophy, although it allows a more dynamic communication patterns' reconfiguration during the system evolution.

The Fishmarket protocol is a multiagent interoperability metaphor characterised by a high degree of interaction, which is essentially placed into the Auction Scene. During the Bidding Round the Auctioneer iteratively sends the Buyers the new prices until he receives the first bid which stops the round. This loop is very fast and usually the Auctioneer performs many sending operations before the round stops.

![Figure 6: The Communication Channel](image)

As a matter of fact, the communication during the Auction Scene is very intensive in one way (from the Auctioneer to the Buyers). Before each sending operation the Auctioneer looks only for bids concerning
we adopted the "locking strategy": once the Auctioneer receives, at the round start, the Buyer list from the Admitter, it assumes that those Buyers will participate in the Bidding Round until its conclusion.

A possible improvement relies on the possibility to avoid this constraint by using the PVM grouping routines. It is possible to organise Buyer programs in a group representing the "current registered Buyers". Each Buyer is able to join or leave this group at any moment, also during the Bidding Round, since the Auctioneer can send price changes to the whole group without being aware of the Buyers number and identifiers. In fact the group server is in charge of sending messages to all group members taking into account the dynamic reconfiguration of the group.

We hope that our Fishmarket simplified implementation built on top of PVM can be used as a test-bed for a more exhaustive and complete implementation which should take into account all the improvements we pointed out. Moreover we think that this work can be considered as a first simple prototype for even more sophisticated multiagent interaction environments [9,10] in distributed settings.

Bibliography


