Downloaded from www.VTUplanet.com

Formalization of Virtual Enterprise using CPL (Case: Electronic Marketplace)

Anoop Kumar Srivastava
Department of Computer Science & Engineering
Radha Govind Engineering College
Anuyogipuram, Garh Road, Meerut-250 004 (U.P.), INDIA
Email: anoop_kumar_srivastava@yahoo.com

***Abstract*—In this paper, we present formalization of Virtual Enterprise with an example of Electronic Marketplace using Concurrent Programming Language (CPL) and provide a multi autonomous agent based framework. Our agent based architecture leads to flexible design of a spectrum of virtual enterprises by distributing computation and by providing a unified interface to data and programs. Autonomous agents are intelligent enough and provide autonomy, simplicity of communication, computation, and a well developed semantics. The steps of design and implementation are discussed. The structure of *Electronic Marketplace*, the agent model, an ontology, interaction pattern between agents and the formalization of *Electronic Marketplace* in CPL is given. We have developed mechanisms for coordination between agents using a language, which we call Virtual Enterprise Modeling Language (VEML). VEML is a dialect of Java and includes Knowledge Query and Manipulation Language (KQML) primitives. We have implemented a multi autonomous agent based system, which we call VE System. VE System provides application programmers with potential to globally develop different kinds of VEs based on their requirements and applications. We demonstrate efficacy of our system by discussing its salient features.**

I. Introduction

With advances taking place in information, transportation, networking and communication technology, companies/factories are being organized as a network of units, each unit corresponding to a well defined objective in a classical set up such as production plant, storage plant, transportation hub, customer relation etc. without the necessity of either locating all the units at the same physical location of the plant or owning all the units by the management of the primary plant establishment. This leads to the description of an *Extended Enterprise* where all resources such as stock, space and production capacity of all the enterprises are added together. Such a real-life model has come to be referred to as *Virtual Enterprises* [1]. A Virtual Enterprise (VE) is an enterprise which has no resource of its own but consists of shared and coordinated activities which utilizes the resources of participating enterprises.

The Virtual Enterprise presents many challenges and opportunities for *Artificial Intelligence* (AI) technology. Adding together the resources of all the participating enterprises of VE stresses technology for information retrieval, networking, communication, coordination, decision making and so on. Satisfying diverse user needs calls for advanced interfaces, user modeling and other emerging techniques. Between users and VE is a gulf filled with a large and evolving network of services that must be selectively arranged to accomplish a particular task.

We argue that a fundamental role for AI in Virtual Enterprise is to perform the underlying monitoring, management, and allocation of services and resources that bring together users and information. While user interfaces and retrieval technologies are important, it is obvious that such technologies will undergo ongoing and dramatic change, which in turn can lead to the restructuring of "how things are done" in VE. In a very real sense, therefore, the underlying architecture of VE must continually search through a growing, shifting, and complex space of content and capabilities for combinations that best serve needs arising at a given moment. This paper aims to satisfy these criteria with an architecture based on distributed software agents. The agents in this system represent specialized information-processing functions, with additional abilities to reason about their effectiveness and requirements, communicate with other agents, negotiate about terms and resources, and perform other generic agent functions. We call them autonomous agents because we assign these modules autonomy of action (they choose what services to perform for whom with what resources under what terms), under the general constraints imposed by their situations and their relations with other agents [7,8,12]. Typical Virtual Enterprise tasks require teaming among numerous specialized agents. Our description so far has been generic, because the scope of what constitutes an "information processing task" relevant to the Virtual Enterprise is broad. To make our conception more concrete, the types of agents that we have built for electronic commerce include:

- **User interface agents:** To manage the presentation of information and input from the user.
- **Information service provider agents:** To perform specific services such as search on databases.
- **Facilitator agents:** To support the location of relevant agents and mediation among them.

To realize this vision, first, we have given the structure of Electronic-Marketplace, next we have provided the agent model, designed an ontology and given interaction pattern

978-1-4244-4711-4/09/$25.00 ©2009 IEEE

Downloaded from www.VTUplanet.com

between agents. The notion of VE is discussed through formalization of Electronic-Marketplace. To demonstrate the viability of the proposed coordination schemes, a programming language is designed and developed, which is called Virtual Enterprise Modeling Language (VEML) and it's run-time architecture to construct the descriptions mutually understood by VE agents [9,10,11]. Knowledge Query and Manipulation Language (KQML) [2,3,4,5,6] which is based on speech act categories for describing protocols and agent communication strategies is used. In particular, we have been able to identify a number of speech acts (and appropriate semantics) that cover a broad range of information services. In addition, we have used speech acts to cover negotiation: the process by which a set of agents come to terms on provision of information services and allocation of resources to the various service activities. The Autonomous Agent Based Virtual Enterprise System which we call VE System has been implemented in VEML. The VE System provides application programmers with potential to globally develop different kinds of VEs based on their requirements and applications. The ultimate goal of VE System is to effectively turn the various participating enterprises into a unified, dependable, secure and distributed computing infrastructure of VE.

This paper is organized as follows. Section 2 deals with design. Section 3 describes the implementation. Section 4 discusses salient features. Section 5 presents conclusion.

II. Design

The design of Electonic-Marketplace is done and structure of E-Marketplace, agent model, an ontology, interaction pattern between agents is provided.

A. An Electronic Marketplace

An Electronic-Marketplace or Digital-Marketplace is a virtual marketplace where buying and selling is done over a communication network. An E-Marketplace comprises entities like buyers, sellers, facilitators and products. The buyers and sellers buy and sell products respectively. Buyers look for information which they are interested in and sellers look for potential buyers who would buy their products. Many a times both the parties spend a lot of time in searching for information about each other. In such a situation a facilitator can act as an intermediary to provide the required information and thereby can ease the process of searching for information. The role of facilitator can be more sophisticated in the sense that it can mediate between buyer and a seller to settle a business deal through a process of negotiation. A number of E-Marketplaces are available ranging from music CDs to automobiles. E-Marketplace dramatically reduces transaction costs and on the other hand can enhance satisfaction of both buyer and seller. E-Marketplace affects the consumer purchase process. It provides a mechanism for reducing the search costs (money, time and effort expended to gather product price, quality and feature information) for consumers. The combination of more information, electronic links and channels will give the buyers more choices, resulting in a shift of bargaining power to the buyers. The product price, search costs, marketing (advertising) costs, overhead costs, inventory costs and production costs are lower. Security is a critical issue in Electronic-Commerce. Consumers demand secure systems if they are to use EC payment.

B. Structure of an Electronic-Marketplace

The structure of E-Marketplace is given below:

- E_Marketplace
  - Agents
    - Buying_agents
      - Registration
        - Direct_Buying
          - Query
          - Advertise
          - Negotiate
        - Indirect_Buying
          - Query
          - Advertise
          - Negotiate
      - Deregistration
    - Selling_agents
      - Registration
        - Direct_Selling
          - Query
          - Advertise
          - Negotiate
        - Indirect_Selling
          - Query
          - Advertise
          - Negotiate
      - Deregistration
    - Facilitators
      - Interaction Buying_agents
        - Registration
        - Query
        - Advertise
        - Deregistration
      - Interaction Selling_agents
        - Registration
        - Query
        - Advertise
        - Deregistration
      - Interaction Facilitators
        - Query
        - Advertise

C. Agent Model

A three tier architecture model of *Electronic Marketplace* (E-Marketplace) [7] is presented consisting of buyers, sellers and facilitators. They interact with each other through their software counterparts called as buying agents, selling agents and facilitators. The facilitators provide various services to buying and selling agents. They are also called software agents or intelligent agents or simply agents. In short software agents are the artifacts which are intended to perform on behalf of their human counterparts. They carry out the tasks without human intervention. Figure 1. represents model of an agent mediated E-Marketplace.

Downloaded from www.VTUplanet.com

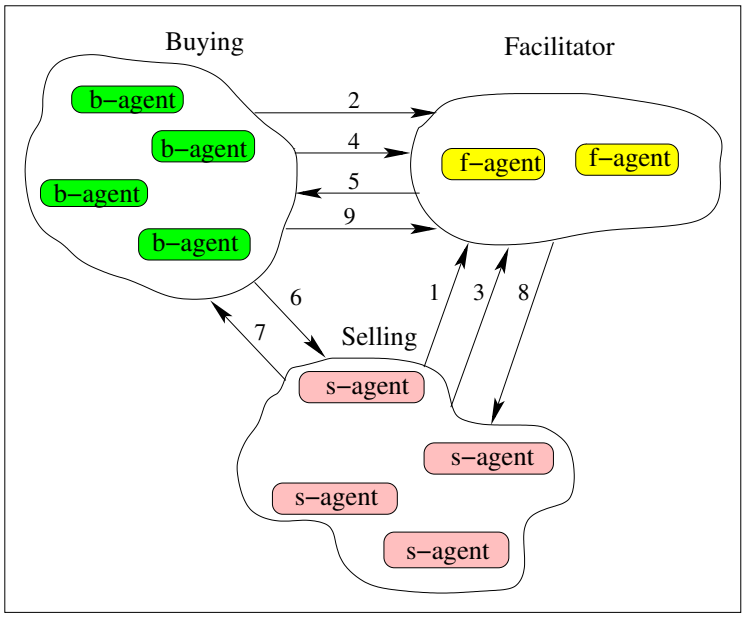

Fig. 1. A model of an agent mediated E-Marketplace

D. Electronic Marketplace Ontology

This is a name which signifies the shared assumptions that the programs have about the knowledge they are using. It is a keyword shared among programs to keep other programs with the same topic from answering.

In the construction of Electronic Marketplace Ontology we followed whenever possible the methodology for developing ontologies outlined by [Uschold et al., 1995b]. This methodology includes the following steps: identify purpose, build the ontology (capture, code, integrate existing ontologies), evaluation and documentation.

E_MARKETPLACE: is a set of inter-related UNITS which are totally committed to some common *PURPOSE* (Buying/Selling).

UNIT: is an entity for *MANAGING* the *ACTIVITIES* to achieve one or more *PURPOSE*. A UNIT may be a buying process, selling process and a facilitator.

BUYING_AGENT: is an entity for performing the activities to *ACHIEVE* the *PURPOSE* of buying.

SELLING_AGENT: is an entity for performing the activities to *ACHIEVE* the *PURPOSE* of selling.

FACILITATOR: is an entity for facilitating the activities to *ACHIEVE* the *PURPOSE* of buying and selling.

PRODUCT: is the item which is purchased or sold in the E_MARKETPLACE.

BUY_REQUEST: is a statement defining a buying agent's needs in terms of PRODUCT, *QUANTITY* and *TIME LIMIT*.

SELL_REQUEST: is a statement defining a selling agent's needs in terms of PRODUCT, *QUANTITY* and *TIME LIMIT*.

AGENT: is an entity for performing the activities to *ACHIEVE* the *PURPOSE* of buying, selling and facilitating.

REGISTRATION: is a process by which a buying agent or selling agent gets entitled to participate in the E_MARKETPLACE.

DEREGISTRATION: is a process by which a buying agent or selling agent are released from the E_MARKETPLACE.

TIME LIMIT: is the period during which the products are due to be purchased or sold.

QUERY: is a process of buying agent and selling agent asking for information about PRODUCT, *PRICE* and *TIME LIMIT*.

NEGOTIATION: is a method by which buying and selling agents will make deal.

ADVERTISE: is a method by which buying and selling agents will make each other aware about their products.

BROADCAST: is a method by which facilitator will send the message to all the buying or selling agents.

DIRECT_BUYING/DIRECT_SELLING: is a method by which buying and selling agents will negotiate with each other directly i.e. without facilitator.

INDIRECT_BUYING/INDIRECT_SELLING: is a method by which buying and selling agents will negotiate with each other through the facilitator.

E. Inter Agent Communication using Performatives

Table I shows communication pattern between agents.

TABLE I
INTERACTION SUMMARY BETWEEN BA, SA and FA

Source	Destination	Message	Message Content	Performative
SA	FA	1	register selling agent with product and parameters	register
BA	FA	2	register buying agent with product and parameters	register
SA	FA	3	advertise selling product with parameters	advertise
BA	FA	4	query buying product	ask-if
FA	BA	5	reply from FA with list of selling agents	reply
BA	SA	6	negotiate price with selling agents	ask-all
SA	BA	7	negotiate price with buying agent	ask-one
BA	FA	8	deregister buying agent	deregister
SA	FA	9	deregister selling agent	deregister

BA represents Buying Agent, SA represents Selling Agent and FA represents Facilitator Agent. These agents communicate through a message called performative. In that the message is

Downloaded from www.VTUplanet.com

intended to perform some action by virtue of being sent. List of reserved performatives used between sender and receiver are e.g. register, advertise, broadcast, ask-if, ask-one, ask-all, reply and unregister etc.

III. Formalization of Electronic-Marketplace

In this section we present formalization of Electronic Marketplace using Concurrent Programming Language (CPL) [13]. The techniques described show how to synchronize the execution of the processes and how to communicate data among them. The language uses the concurrent programming tools called processes and monitors. There are three concurrent processes running in E-Marketplace e.g. buying, selling and facilitator. Buying, selling, and facilitator interact with each other. "reception" is a data structure shared by three concurrent processes. All the processes need to know that they can send and receive data through it. This kind of system component is called a monitor. A monitor can synchronize concurrent processes and transmit data between them.

A. E-Marketplace

```
program E_Marketplace;
type
    reception1 = monitor;
    reception2 = monitor;
    buyingprocess = process( b_f_receptionist,
b_s_receptionist:
    reception1; b_b_receptionist: reception2 );
    sellingprocess = process( s_f_receptionist,
b_s_receptionist:
    reception1; b_b_receptionist: reception2 );
    facilitatorprocess = process( b_f_receptionist,
    s_f_receptionist: reception1 );
var
    b_f_receptionist, s_f_receptionist,
b_s_receptionist:reception1;
    b_b_receptionist : reception2;
    Buying: buyingprocess;
    Selling: sellingprocess;
    Facilitator: facilitatorprocess;
begin
     init  b_f_receptionist,
           s_f_receptionist,
           b_s_receptionist,
           b_b_receptionist,
           Buying( b_f_receptionist,
b_s_receptionist,
           b_b_receptionist ),
           Selling( s_f_receptionist,
b_s_receptionist,
           b_b_receptionist ),
           Facilitator( b_f_receptionist,
s_f_receptionist );
end
```

We have defined reception as a monitor. There are two monitor procedures, send and receive. Processes can not operate on shared data. They can only call monitor procedures that have access to the shared data.

B. Selling agent

Registration

```
    type sellingprocess = process( s_f_receptionist,
b_s_receptionist:
                  reception1; b_b_receptionist:
reception2 );
  var  selling_agent, selling_product: array [ 1..32
] of char:
       no_of_buying_agent: integer;
     procedure  register_selling_agent(  selling_agent,
selling_product );
begin
     s_f_receptionist.send( selling_agent,
selling_product );
     s_f_receptionist.receive( acknowledgement );
end
```

register_selling_agent(selling_agent, selling_product): The selling agent sends register request to facilitator with agent name and product name. The facilitator receives the request and registers selling agent with selling product and then updates its database. The facilitator sends acknowledgement to the selling agent. The selling agent can follow two types of selling methods e.g. direct_selling and indirect_selling.

```
procedure query_selling_product( selling_product );
begin
     b_b_receptionist.read( selling_product );
end
procedure advertise_selling_product( selling_product );
begin
        b_b_receptionist.write( selling_agent,
selling_product );
end
procedure negotiate_selling( no_of_buying_agent );
var selling_price: integer;
     acknowledgement: array [ 1..32 ] of char;
begin
      agent_count = 0;
      repeat
      b_s_receptionist.send( selling_price );
      b_s_receptionist.receive( buying_price );
      agent_count = agent_count + 1;
      until agent_count = no_of_buying_agent;
      b_s_receptionist.send( selling_price );
      b_s_receptionist.receive( acknowledgement );
end
```

Direct_Selling: In direct selling the selling agents are open to buying agents. The selling agents do not interact with the buying agents through facilitator. The selling agents follow three procedures.

query_selling_product(selling_product): The selling agent queries about the product by reading from billboard. The selling agent will read the agents buying the selling product from the billboard with the help of billboard receptionist. This will help in negotiating the price with the list of buying agents.

advertise_selling_product(selling_product): The selling agent advertises the selling product by writing the name of the product on billboard.

negotiate_selling(no_of_buying_agent): The selling agent negotiates the price with all the shortlisted buying agents one by one. It sends the selling price to individual buying agent

Downloaded from www.VTUplanet.com

and receives the buying price from buying agent. This will be repeated till all the selected no. of buying agents are contacted. Finally, the price will be mutually agreed by the buying and selling agent and deal will be made. The final price is received by buying agent and the buying agent sends acknowledgement to the selling agent.

Indirect_Selling: In indirect selling the selling agents are anonymous to buying agents. The selling agents interact with the buying agents through facilitator. The selling agents follow three procedures.

query_selling_product(selling_product): The selling agent sends query about product to facilitator. The facilitator sees its database and finds out the list of buying agents for the product and sends the list to selling agent.

advertise_selling_product(selling_product): The selling agent sends the request of advertisement to facilitator. The facilitator will send the request to all the buying agents.

negotiate_selling(no_of_buying_agent): The selling agent negotiates with all the short listed buying agents one by one. It sends selling price to the individual buying agent and receives the buying price from buying agent. This is repeated till all the selected no. of buying agents are contacted. Finally, the price is mutually agreed by the buying and selling agent and deal is made. The final price is received by the buying agent and the buying agent sends the acknowledgement to selling agent.

```
 procedure query_selling_product( selling_product );
begin
      s_f_receptionist.send( selling_product );
      s_f_receptionist.receive( agentlist );
end
 procedure advertise_selling_product( selling_product );
begin
      s_f_receptionist.send( selling_agent,
selling_product );
end
 procedure negotiate_selling( no_of_buying_agent );
 var selling_price: integer;
     acknowledgement: array [ 1..32 ] of char;
begin
      agent_count = 0;
      repeat
      b_s_receptionist.send( selling_price );
      b_s_receptionist.receive( buying_price );
      agent_count = agent_count + 1;
      until agent_count = no_of_buying_agent;
      b_s_receptionist.send( selling_price );
      b_s_receptionist.receive( acknowledgement );
end
```

Deregistration

```
   procedure deregister_selling_agent( selling_agent,
selling_product );
begin
      s_f_receptionist.send( selling_agent,
selling_product );
      s_f_receptionist.receive( acknowledgement );
end
cycle
      selling_agent = àmit; selling_product =
čamera;
      register_selling_agent( selling_agent,
selling_product );
      query_selling_product( selling_product );
      advertise_selling_product( selling_product );
      no_of_buying_agent = n;
      negotiate_selling( no_of_buying_agent );
      deregister_selling_agent( selling_agent,
selling_product );
end
```

deregister_selling_agent(selling_agent, selling_product): The selling agent sends the deregister request to facilitator with the agent name and product name. The facilitator deregisters the selling agent with the product and then send acknowledgement to the selling agent.

C. Buying agent

Registration

```
     type buyingprocess = process( b_f_receptionist,
b_s_receptionist:
                  reception1; b_b_receptionist:
reception2 );
   var  buying_agent, buying_product: array [ 1..32 ]
of char:
                no_of_selling_agent: integer;
      procedure register_buying_agent( buying_agent,
buying_product );
begin
      b_f_receptionist.send( buying_agent,
buying_product );
               b_f_receptionist.receive( acknowledge-
ment );
end
```

register_buying_agent(buying_agent, buying_product): The buying agent sends register request to facilitator with agent name and product name. The facilitator receives the request and registers buying agent with buying product and then updates its database. The facilitator sends acknowledgement to the buying agent. The buying agent can follow two types of buying methods e.g. direct_buying and indirect_buying.

```
 procedure query_buying_product( buying_product );
begin
              b_b_receptionist.read( buying_product
);
end
 procedure advertise_buying_product( buying_product );
begin
        b_b_receptionist.write( buying_agent,
buying_product );
end
 procedure negotiate_buying( no_of_selling_agent );
 var buying_price: integer;
     acknowledgement: array [ 1..32 ] of char;
begin
               agent_count = 0;
               repeat
               b_s_receptionist.send( buying_price
);
               b_s_receptionist.receive(
selling_price );
               agent_count = agent_count + 1;
               until agent_count =
no_of_selling_agent;
```

Downloaded from www.VTUplanet.com

```
                b_s_receptionist.send( buying_price
);
                b_s_receptionist.receive(
acknowledgement );
  end
```

Direct_Selling: In direct selling the selling agents are open to buying agents. The selling agents do not interact with the buying agents through facilitator. The selling agents follow three procedures.

query_buying_product(buying_product): The buying agent queries about the product by reading from billboard. The buying agent will read the agents selling the buying product from the billboard with the help of billboard receptionist. This will help in negotiating the price with the list of selling agents.

advertise_buying_product(buying_product): The buying agent advertises the buying product by writing the name of the product on billboard.

negotiate_buying(no_of_selling_agent): The buying agent negotiates the price with all the shortlisted selling agents one by one. It sends the buying price to individual selling agent and receives the selling price from selling agent. This will be repeated till all the selected no. of selling agents are contacted. Finally, the price will be mutually agreed by the buying and selling agent and deal will be made. The final price is received by selling agent and the buying agent sends acknowledgement to the selling agent.

Indirect_Selling: In indirect selling the selling agents are anonymous to buying agents. The selling agents interact with the buying agents through facilitator. The selling agents follow three procedures.

query_buying_product(buying_product): The buying agent sends query about product to facilitator. The facilitator sees its database and finds out the list of selling agents for the product and sends the list to selling agent.

advertise_buying_product(buying_product): The selling agent sends the request of advertisement to facilitator. The facilitator will send the request to all the buying agents.

negotiate_buying(no_of_selling_agent): The buying agent negotiates with all the short listed selling agents one by one. It sends buying price to the individual selling agent and receives the selling price from selling agent. This is repeated till all the selected no. of selling agents are contacted. Finally, the price is mutually agreed by the buying and selling agent and deal is made. The final price is received by the buying agent and the buying agent sends the acknowledgement to selling agent.

```
 procedure query_buying_product( buying_product );
begin
      b_f_receptionist.send( selling_product );
                b_f_receptionist.receive( agentlist
);
end
 procedure advertise_buying_product( buying_product );
begin
                s_f_receptionist.send( buying_agent,
buying_product );
end
 procedure negotiate_buying( no_of_selling_agent );
 var buying_price: integer;
     acknowledgement: array [ 1..32 ] of char;
begin
      agent_count = 0;
              repeat
              b_s_receptionist.send( buying_price
);
      b_s_receptionist.receive( selling_price );
              agent_count = agent_count + 1;
              until agent_count =
no_of_selling_agent;
              b_s_receptionist.send( buying_price
);
              b_s_receptionist.receive(
acknowledgement );
end
```

Deregistration

```
  procedure deregister_buying_agent( buying_agent,
buying_product );
begin
              b_f_receptionist.send( buying_agent,
buying_product );
              b_f_receptionist.receive(
acknowledgement );
end
cycle
      buying_agent = àmit; buying_product = ċamera;
      register_buying_agent( buying_agent,
buying_product );
              query_buying_product( buying_product
);
              advertise_buying_product(
buying_product );
              no_of_selling_agent = n;
              negotiate_buying( no_of_selling_agent
);
      deregister_buying_agent( buying_agent,
buying_product );
end
```

deregister_selling_agent(selling_agent, selling_product): The selling agent sends the deregister request to facilitator with the agent name and product name. The facilitator deregisters the selling agent with the product and then send acknowledgement to the selling agent.

D. Facilitator

The facilitator receives the register request from buying/selling agent. It registers the buying/selling agent and sends the acknowledgement as registered to the respective buying/selling agent and then update its database.

Registration

```
type facilitatorprocess = process( b_f_receptionist,
                        s_f_receptionist: reception1
);
 var  buying_agent, selling_agent, buying_product,
      selling_product: array [ 1..32 ] of char;
    procedure  register_buying_agent(  buying_agent,
buying_product );
 var  acknowledgement: array [ 1..32 ] of char;
begin
```

Downloaded from www.VTUplanet.com

```
        b_f_receptionist.receive( buying_agent,
buying_product );
        register( buying_agent, buying_product );
        acknowledgement = ṙegistered;
        b_f_receptionist.send( acknowledgement );
        update( buying_agent, buying_product );
  end
      procedure register_selling_agent( selling_agent,
selling_product );
   var acknowledgement: array [ 1..32 ] of char;
  begin
       s_f_receptionist.receive( selling_agent,
selling_product );
       register( selling_agent, selling_product );
       acknowledgement = ṙegistered;
       s_f_receptionist.send( acknowledgement );
       update( selling_agent, selling_product );
  end
```

query_buying_product(buying_product); query_selling_product(selling_product): The facilitator receives query request fron buying/selling agent about their product. It sees its database and acquire the list of buying/selling agents and sends it to buying/selling agent respectively.

Query-product

```
   procedure query_buying_product( buying_product );
   var agentlist: array [ 1..512 ] of char;
  begin
       b_f_receptionist.receive( buying_product );
       acquire( buying_product );
       agentlist = list_of_selling_agents;
       b_f_receptionist.send( agentlist );
  end
   procedure query_selling_product( selling_product );
   var agentlist: array [ 1..512 ] of char;
  begin
       s_f_receptionist.receive( selling_product );
       acquire( selling_product );
       agentlist = list_of_buying_agents;
       s_f_receptionist.send( agentlist );
  end

   procedure advertise_buying_product( buying_product );
  begin
       b_f_receptionist.receive( buying_product );
       s_f_receptionist.send( buying_product );
  end
   procedure advertise_selling_product( selling_product );
  begin
       s_f_receptionist.receive( selling_product );
       b_f_receptionist.send( selling_product );
  end
```

Advertise-product:

advertise_buying_product(buying_product); advertise_selling_product(selling_product): The facilitator receives the request of advertisement from buying/selling agent about their product. It accordingly sends the request to all the buying/selling agents respectively.

Deregistration

```
    procedure deregister_buying_agent( buying_agent, buy-
ing_product );
   var acknowledgement: array [ 1..32 ] of char;
  begin
       b_f_receptionist.receive( buying_agent,
buying_product );
       deregister( buying_agent, buying_product );
       acknowledgement = ḍeregistered;
       b_f_receptionist.send( acknowledgement );
       update( buying_agent, buying_product );
  end
    procedure deregister_selling_agent( selling_agent, sell-
ing_product );
   var acknowledgement: array [ 1..32 ] of char;
  begin
       s_f_receptionist.receive( selling_agent,
selling_product );
       deregister( selling_agent, selling_product );
       acknowledgement = ḍeregistered;
       s_f_receptionist.send( acknowledgement );
       update( selling_agent, selling_product );
  end
```

deregister_buying_agent(buying_agent, buying_product); deregister_selling_agent(selling_agent, selling_product): The facilitator receives the deregister request from buying/selling agent. It deregisters the buying/selling agent and sends the acknowledgement as deregistered to respective buying/selling agent and then updates its database.

```
  cycle
                  /*  For buying agent  */
       buying_agent = ànoop; buying_product =
ċamera;
       register_buying_agent( buying_agent,
buying_product );
       query_buying_product( buying_product );
       advertise_buying_product( buying_product );
       deregister_buying_agent( buying_agent,
buying_product );
                   /*  For selling agent */
       selling_agent = àmit; selling_product =
ċamera;
       register_selling_agent( selling_agent,
selling_product );
       query_selling_product( selling_product );
       advertise_selling_product( selling_product );
       deregister_selling_agent( selling_agent,
selling_product );
  end

   type reception1 = monitor;
       var      receptionboard: board; sender_agent,
receiver_agent: queue;
       message: string;
   procedure send;
  begin
      receptionboard.write( message );
      continue( receiver_agent );
  end
   procedure receive;
  begin
      receptionboard.read( message );
      continue( sender_agent );
  end
  begin
      init receptionboard;
  end
```

Downloaded from www.VTUplanet.com

```
 type reception2 = monitor;
 var  billboard: board; message: string;
 procedure read;
begin
      billboard.request;
      billboard.read( message );
      billboard.release;
end
 procedure write;
begin
      billboard.request;
      billboard.write( message );
      billboard.release;
end
begin
      init billboard;
end
```

IV. Implementation

To test the validity of what has been described, a VE System is implemented, which demonstrates electronic commerce. We have designed a language called Virtual Enterprise Modeling Language (VEML) and its methods. The application programs are written in VEML which get preprocessed into Java programs by our VEML compiler. Java program uses agent library and get eventually compiled into Java byte code, for execution atop any Java Virtual Machine (JVM).

A. Virtual Enterprise Modeling Language (VEML)

VEML is a dialect of the Java programming language directly enabling software agent oriented programming and development of VE [15]. VEML includes KQML primitives. KQML is a language and an associated protocol to support the high level communication between autonomous agents. KQML is an abstraction, a collection of primitives plus the assumption of a simple model for inter-agent communication. There is no such thing as an implementation of KQML in the sense that KQML is not a compiled language. VEML is a compiled language. The VEML grammar includes Java grammar with additional keywords and statements.

VEML is a tool to build software applications that dynamically interact and communicate with their immediate environment (user, local resources and computer system) and/or the world, in an autonomous (or semi-autonomous), task oriented fashion. VEML simplifies programming on the Internet by providing synchronous and asynchronous communication among agents with the help of message passing. Universal naming is possible in VEML. VEML supports communication among multiple inter generating computations/nodes [14]. It is a language for programs to use to communicate attitudes about information, such as querying, stating, achieving, believing, requiring, subscribing and offering.

B. Architecture

Figure 3. shows the architecture of VE System, which comprises the following components: (a) Front-end: a GUI handles all of the user interactions. The front end is implemented in VEML. Front end is a screen shot of VE showing the form, which user has to fill for different purposes. (b) Back-end: a virtual enterprise engine where the agents actually "live" and interact with one another. The back end is implemented in VEML and KQML. The back-end implements a request response service: a client sends a request to the back-end, the back-end services that request. The front-end and back-end communicates with one other via TCP/IP sockets.

To test the validity of what has been described, we have designed and implemented a VE System (An Automated Multi Intelligent Based System) which demonstrates the example of Electronic-Marketplace. The distinguished features are as follows:

- This is an automated multi intelligent agent based system.
- This system automates the process of buying and selling goods. The users can create autonomous agents to buy and sell goods on their behalf.
- The agents will automatically negotiate and make the best possible deal on user's behalf.
- The buying/selling agents are pro-active. They try to buy/sell themselves by going into the marketplace contacting buying/selling agents and negotiating with them to find the best deal.
- This system eliminates human-human contact.
- The agents can be run parallel.
- The following parameters are supported by the system for Electronic-Marketplace:
  - Desired period to sell the item by: People usually have a deadline by which they want to sell something.
  - Desired price: This is the price the user would like to sell their good for.
  - Lowest acceptable price: This is the lowest price the user will sell their good for.
  - Highest acceptable price: This is the highest price the user will buy the good for.
  - Desired physical region: This is the region selected by the user for the item to be delivered by seller.
  - Decay function of price (Linear, Quadratic and Cubic): The user has some control over the agent's negotiation "strategy". The user can specify the "decay" function the agent uses to lower the asking price over its given time frame. The user has three choices: linear, quadratic and cubic.

V. Salient Features

The salient features of VE System are as follows:

1) **Naming, Security, and Direct Connection:** Agents can have any number of logical names that don't contain the hostname. There is no way for an agent to "overhear" a conversation between two other agents. This provides security. Agents can establish direct connections with each other for bulk data transfer.
2) **Third Party Creation of Agents:** We have used protocols and semantics for defining more extensible agent communications. "This protocol is open to allow third-party agents (with their own unique strategies)

Downloaded from www.VTUplanet.com

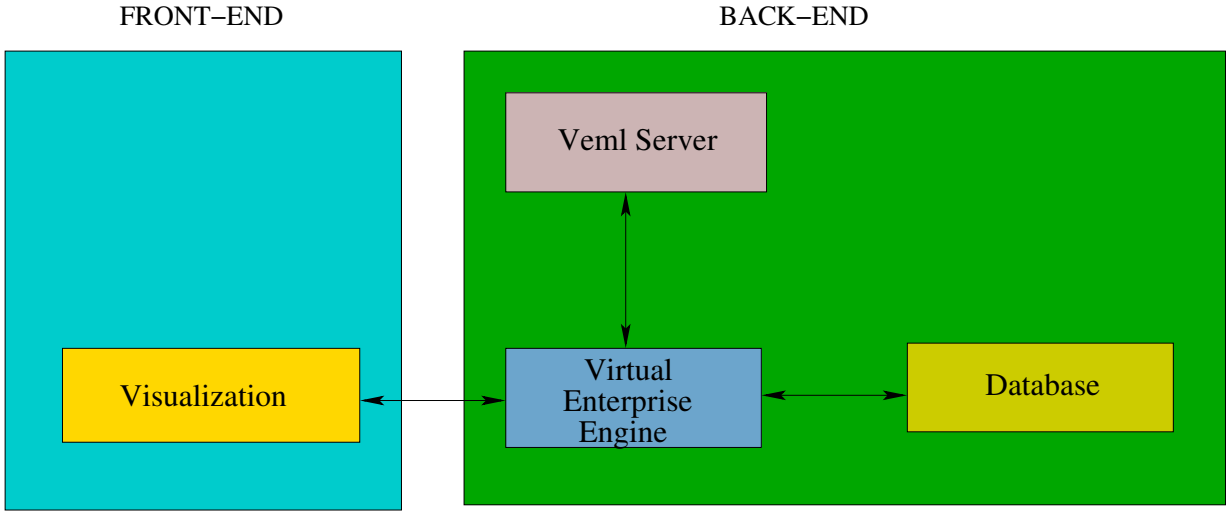

Fig. 2. Architecture of Virtual Enterprise System

to participate in the agent coordination". We envision developers creating sophisticated commerce agents that require a non-trivial amount of resources to complete it's market analysis. Such agents could potentially be run more efficiently on a user's local machine and communicate/negotiate with other agents via open protocol. Agents can be developed in any language e.g. C, Lisp, Prolog, Java and VEML. This system supports protocol like TCP/IP, SMTP, and FTP.

3) **Routers:** The router in this system provides an easy-to-use link between application and network viz. (1) Routers are a content independent message routers, (2) All routers are identical, just an executing copy of the same program, (3) A router handles all messages going to and from it's associated agent i.e. each agent has it's own separate router process. Thus it is not necessary to make extensive changes in the program's internal organisation to allow it to asynchronously receive messages from a variety of independent sources, (4) The router provides this service for the agent and provides the agent with a single contact point for the communication with the rest of the network, (5) It provides both client and server functions for the application and can manage multiple simultaneous connections with other agents, (6) Routers relies solely on its performatives and arguments, (7) A router directs a message to a particular Internet address as specified by the message, (8) When an application exits, the router sends another message to the facilitator, removing the application from the Facilitator's database, and (9) Routers can be implemented with varying degree of sophistication although they can guarantee to deliver all messages.
4) **Facilitator:** The facilitator agent of this system performs following useful services: (1) Maintain a registry of service names, (2) Forward message to named services, (3) Routes messages based on the content, (4) Provides matchmaking between information providers and clients, (5) Provides mediation and translation services, (6) This is a simple software agent which maintains the database of active agents, (7) This provides the registry of agent names and addresses, (8) It is used by all the agents as they arise to register their names and addresses and to subsequently locate other agents to which messages are to be sent, and (9) It accepts register and unregister performatives to maintain it's database and responds to ask-one and ask-all etc.
5) **Scalability:** When an agent is created, it acquires a Java thread. So, depending on the number of agents created, it acquires that many number of threads. These threads are software programs. Each thread defines a separate path for execution. Multi tasking threads require less overhead than multi tasking processes. Processes are heavyweight tasks that require their own separate address spaces. Interprocess communication is expensive and limited. Context switching from one process to another is also costly. Threads on the other hand are lightweight. They share the same address space and cooperatively they share the same heavyweight process. Interthread communication is inexpensive and context switching from one thread to the next is low cost.
6) **Performance and Efficiency:** The system performance guarantees to agents so that the agents can meet real time constraints. The parameters of argument are time to complete the process, cost factor, bandwidth requirement and transfer of code. This system is a single unified framework with message passing in which wide range of distributed applications can be implemented efficiently and easily. This system uses a network service by remotely invoking its operations. The results are transmitted back over the network to the sending agent which processes them depending on the result.

Downloaded from www.VTUplanet.com

VI. Conclusion

This paper presents formalization of Virtual Enterprise using with an example of Electronic Marketplace using CPL. We have provided structure, an ontology, agent model, interaction pattern between agents and formalization. We have developed an agent oriented virtual enterprise modeling language (VEML) and have provided the methods used for working of the agents. Finally, the architecture and salient features of VE system are provided. Here our focus is in supporting the best communication, co-ordination and problem solving mechanism available with minimum programming effort on the developer's side. Hence this architecture can be directly applied in practice.

This paper also presents an application of AI to the implementation of Virtual Enterprise which is a multi autonomous agent based system. As all the facts surrounding software agents recommends, Virtual Enterprises are just one area for a new generation of automated information services. AI has a large role to play in improving the generality, robustness and overall competence of these services. We believe that the field of AI has an equally important role to play in architectural infrastructure (in concert with other technologies, of course). In a Virtual Enterprise, our typical problem is an abundance of available information and information services. Efficiently bringing together the right agents with the right resources for the right tasks is the measure of the VE's effectiveness.

Acknowledgement

I would like to acknowledge Mr. Yogesh Tyagi, Chairman, Shree Krishna Shiksha Prasar Samiti for providing all kinds of support to carry out this research work and being a constant source of inspiration.

References

[1] Luis M. Camarinha and Hamideh Afsarmanesh, Virtual Enterprise Modeling and Support Infrastructures: Applying Multi-agent Systems Approaches, In M.Luck et al., editor, ACAI 2001, LNAI 2086, pages 335-364, Springer-Verlag, 2001.

[2] Tim Finin, Don McKay and Rich Fritzson, An Overview of KQML: A Knowledge Query and Manipulation Language, Technical report, Department of Computer Science, University of Maryland, Baltimore County, March 1992.

[3] Lockheed Martin C^2 Integration Systems, 590, Lancaster Ave., Frazer, PA 19355-1808. *Software User's Manual for KQML*, August 1997.

[4] Software Design Document for KQML. Technical report, Unisys Corporation, 70 East Swedesford Road, Paoli, PA 19301, March 1995.

[5] James Mayfield, Yannis Labrou, and Tim Finin, Desiderata for Agent Communication Language, In *proceedings of the 1995 AAAI Spring Symposium on Information Gathering in Distributed Environments*, March 1995.

[6] Tim Finin, Rich Fritzson, Don McKay and Robin McEntire, KQML-An Information and Knowledge Exchange Protocol, In *Knowledge Building and Knowledge Sharing*, Ohmsa and IOS Press, 1994.

[7] Jeffrey M. Bradshaw, An Introduction to Software Agents, In J M Bradshaw, editor, *Software Agents*, MIT Press, 1996.

[8] M. Shaw, R. Blanning, T. Strader and A. Whinston, editors, *Handbook on Electronic Commerce*, chapter1. Springer Verlag, 2000.

[9] A K Srivastava, Intelligent Agent Based Virtual Enterprise System, In Poster proceedings of 24th SGAI AI 2004, Queens' College, Cambridge, U.K. 13-15 Dec.2004.

[10] A K Srivastava, Simulation of a Multi Intelligent Agent Based System, Proceedings of 8th International Conference on Computer Modelling and Simulation UKSim 2005, April 2005,St. John's College, Oxford, UK.

[11] A K Srivastava, Simulation of Virtual Enterprises: A Multi Intelligent Agent Based System, International Journal of Simulation System, Science and Technology (IJSSST), Vol. 7, Oct. 2005.

[12] A K Srivastava, An Application of Artificial Intelligence to the Implementation of Virtual Automobile Manufacturing Enterprise, In Proceedings of 25th SGAI AI 2005, Peterhouse College, Cambridge, U.K., Dec. 12-14 2005.

[13] P. Brinch Hansen, The Programming Language Concurrent Pascal, IEEE Transaction on Software Engineering 1, 2 (June 1975), 199-207.

[14] Katia P. Sycara, Multiagent Systems, AI Magazine, pages 79-92, AAAI Publication 1998.

[15] A K Srivastava, VEML: A Language for Developing Multi Agent Systems, In A K Srivastava, editor ICSC 2008, pages 281-294i, Narosa Publication 2008.