A REAL-TIME AUCTION SYSTEM OVER THE WWW

Christos Bouganis, Dimitrios Koukopoulos and Dimitrios Kalles
Computer Technology Institute
PO Box 1122
GR 261 10
Patras, Greece
E-mail: kalles@cti.gr

KEYWORDS: Computer Software, Finance, Marketing, Real-time simulation

ABSTRACT

This paper presents an electronic auction system as a case study of a distributed computing application using the Java language and the Voyager distributed programming platform. It stresses the importance of state-of-the-art WWW development tools that address the run-to-the-market need of electronic commerce applications. Conceptual simplicity and efficient design and implementation are the major advantages from the adoption of a platform where object mobility and persistence are transparently available.

INTRODUCTION

The WWW hosts some systems of electronic auctions, but these are not real time systems. Typical auctions last several days and it is at their conclusion that a bidder is informed whether he/she was successful or not.

A typical electronic auction systems is Bonsai (www.bonsai.com). In Bonsai new members are registered by filling in a special form. When a user becomes a member in the system he/she can observe an auction, participate in an auction or sell one or more products. To sell a product he/she completes a form, giving information about the product and the desirable (minimum) price. Bids are collected for a pre-specified time period, and bidders are informed about the biggest offer and the final winner. Usually an auction takes quite some time to complete. Similar systems are Jerome J. Manning & Co.:Real Estate Auctions (www.jjmanning.com) and Copart Salvage Auto Auctions (www.copart.com). However, the flourishing of the real-time auction concept has been most pronounced by the flotation of eBay (www.ebay.com).

These systems are not real time however. So, persons or companies who would like to participate in auctions over the WWW, but have limited time resources to do so, can not use them (in reality, some pseudo-autonomous bidding is allowed but this does not compensate for the absence of real time action). A similar problem appears with sellers, who may want their product to be sold as fast as possible (for example an airline may want to auction the remaining seats of a flight a few hours prior to departure). The lack of the above feature is a major problem of these systems and could be a factor of unreliability due to the possibility of generating suspicion that bids are maybe manipulated. The need for a real time electronic auction system is existent.

Another electronic auction system is CASS (http://nas.is/rsf). Even though, CASS is not implemented over the WWW, it is designed as a multi-location real-time electronic marketplace (currently used only in wholesale fish auctions, it provides for the connection of remote sites, such as fish auctions, buyers and sellers, using a PC via a modem).

Network bandwidth (or the lack of it) has always been a limiting factor in the deployment of real time applications over the WWW. However, as the information society will be increasingly dependent on the ability of its citizen to communicative efficiently on-line, the development of products and services that attempt to emulate real life situations is a necessity. Thus, a real-time auction system would exactly address the transition to an electronic consumer habit as close as possible to the conventional one.

The rest of this paper is organised as follows: First an overview of the implemented system is presented along with its key technological components (Java, Voyager, Applets and the WWW). Then, the users' requirements of an on-line electronic auction system and a scenario of usage are specified. Finally, the system design is described.

BRIEF SYSTEM OVERVIEW

The auction system has three types of entities: buyers, sellers and auction houses. It implements a classic scenario: a seller informs an auction house about products to be sold and the auction house forwards them to a group of buyers inviting them to start bidding. The system has the following features:

- a communication protocol between basic processes (an auction house is informed by sellers for the products they want to sell, buyers make offers for products which are auctioned in the auction house, the auction house determines the winner of every auction and informs corresponding sellers about that).
- a facility for recording information about auctions which have been completed, in a suitable database. All transactions which take place in an auction are recorded in a database, so a buyer has the opportunity to see some information on history of bids.
- a process for the authentication of users.
- a graphical user interface for any type of interaction with the system.

The system has a dynamics of expansion. Because of its use of the Voyager technology for the development of the system, it can easily be expanded to incorporate features of artificial intelligence and total independence of the machine in which it runs. This can happen with the use of agents (special types of objects, implemented and made available by Voyager). Agents can move autonomously by embedding onto them routes between auction houses.

Using the Voyager feature of object mobility, an auction house can manage system overload situations by transferring an auction to another computer, in a manner totally transparent to its users. Also, the auction house workload, which corresponds to the messages exchanged between an auction house and its customers, can be reduced with the use of the feature of Space, an advanced architectural feature of Voyager. Using the idea of Space an auction house can divide its users in domains and inform them via domain forwarders transparently, rather than individually.

Java & Voyager

Java is a powerful programming language built to be secure and cross-platform. It is being continuously extended to provide language features and libraries that elegantly handle problems that are difficult in traditional programming languages, such as multithreading, database access, network programming and distributed computing.

Java accommodates client-side programming via the applet (our system uses applets to interact with users).

For the development of the auction system we used ObjectSpace Voyager Core technology, which is a product of ObjectSpace (www.objectspace.com). Voyager, a simple but dynamic object request broker (ORB: a platform/program, which gives the opportunity to programmers to develop distributed applications, facilitating the exchange of messages between objects), is developed on this technology. Voyager is used for the development of distributed applications. It is 100% Java and it follows the object-oriented model of Java. With Voyager a programmer can create remote objects easily, send them messages and move them between programs, which are located in different computers.

Except from the fact that Voyager has a set of features, which are shared by other ORBs too, it also provides mobile objects and autonomous agents. Voyager is a platform which unifies distributed programming with agent technology. Its basic philosophy is that an agent is simply a special type of object, which can move independently, continue to do operations, as it moves and acts like any other object. Voyager permits objects and agents to send Java messages to an agent even if it moves. Also it permits remote activation of any Java class, even if it belongs in a third party library, without modification of the class source. It also supports services such as transparent distributed persistence, scalable group communication and basic operations of catalog administration.

Voyager, unlike other distributed technologies, uses the Java language as an essential interface. A main feature of Java is the ability of class loading in a virtual machine at run-time, permitting the usage of distributed objects and autonomous agents as a tool for the development of distributed systems. Accommodating this feature in older distributed technologies is impractical and often leads to awkward usage of mobile objects and agents, making them clearly inferior to the transparent support of these features in Voyager.

REQUIREMENTS SPECIFICATIONS

Basic user requirements (most of which hold for conventional auctions too) are set out below:

Friendly user interface. The system must provide a user friendly interface to accommodate inexperienced users without intimidating or disappointing them.

Fast access to the auctions' system. Users should not waste time waiting for a connection. Delays may result in a user missing an auction and, surely, are a source of annoyance.

Minimal user involvement in system installation and operation.

Small cost for purchase and usage of the system. The system must be inexpensive to tempt users and auction houses to use it. Although the license price can be fixed ad hoc, the requirements of the system in hardware mustn't be big or cause network overload. Both could either increase costs or delay access, resulting in potential market failure.

Provision of equal opportunities for participation to users. Every subscriber of the system must have the same opportunities of participation in auctions with other users. No privileges (such as early information) are to be given to any users.

Observation and participation in more than one auction. A user must have the opportunity to participate in all currently ongoing auctions, switching between them at will.

Autonomous bids. The potential of autonomous bidding should be explored. Autonomy may be either specified by an ad-hoc schedule or learned via profiling.

Historical information. Users should be able to keep track of an auction's history before deciding to submit a bid.

Notification to users about potential auction conclusion in which they are about to win. Users should be alerted for auctions which are about to finish and in which they will be the winners, if someone doesn't bid for more.

Security. The system must be safe in data exchange, to avoid data perusal by malicious users or plain hackers. Also it must secure that no bid information will be lost. It is also essential that the system supports the privacy of personal information (such as credit card, and/or address details).

Authentication. Corresponding to user identification, it means that every user who participates in any auction is known to the system.

SCENARIO OF USAGE

In this section we will describe the actions available to a user. Registration, which is essential for selling and buying products, is such an action. Product declaration is another action where a seller submits an item to be auctioned. Participation in an auction is another one. Finally, simple observation of the evolution of an auction is a commonplace action, which allows the user to simply observe an auction without the right to make an offer.

Registration

A simple registration form containing only a log-in and a password field is the bare minimum required of an authentication process. To allow maximum separation between classes of users, buyers and sellers are registered separately

Product declaration

Registered sellers submit products for sale specifying the expected (minimum) price, a product description and a date by which the auction must have been completed. The auction house then informs the potential seller if the product is accepted.

Participating in an auction

A buyer participates in an auction by simply accessing the WWW site of the auction house and being granted an authentication clearing.

Observation of an auction

Any user may observe the carrying out of an auction, even without registering. These users are all potential customers.

Expanding the scenarios

The operation of the system was described from the point of view of one auction house only. The descriptions set out above hold when the world consists of one auction house or when auction houses are not co-operating.

The system can be expanded in the situation that some auction houses decide to co-operate. Then, a large space of auction houses will be created, where a user with or without the help of agents could take advantage of several services made available by such an architecture of collaboration. Among such services could be:

- A user can specify his/her requirements for a product (product description, price) and an agent is spawned to locate an auction house, where such an item is being auctioned. The agent may then bid on behalf of the user.

- A seller could achieve less charging from an auction house for auctioning his/her product, as he/she could select among a variety of auction houses.

- Auction houses also stand to reap benefits from such a co-operation. A larger audience (of potential users) will be addressed. This happens because a user of an auction house could be automatically a user of another auction house too.

- An auction house could delegate some of its functionality to a competitor/collaborator, should it not be available for a variety of reasons (e.g. system maintenance - for example, this would work much like the airline industry where competitor company jets are hired to tackle extraordinary situations, such as strikes, increased demand etc).

The system to be presented in this paper would need quite some technical enhancements regarding functionality to achieve such a co-operation platform, but due to the technologies it adopts this would entail minor changes in the conceptual design.

CONCEPTUAL DESIGN

Figure 1 details the system design in OMT notation.

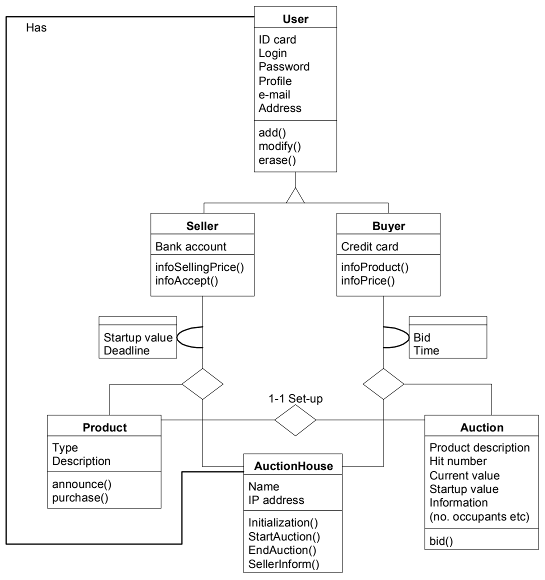

Figure 1: Conceptual design.

The main object of the system is the AuctionHouse. All the other objects hook on it at some point in their lifetime. Its job is mainly to create and to synchronize the other objects. It can initialize and terminate an auction and inform a seller about the acceptance of a product.

The users of the system can be separated in two categories: the sellers and the buyers. Sellers (Seller object) are the users who submit to the system a product to be auctioned. Buyers (Buyer object) are the users who participate in auctions. These two objects are formed by specializing a third object, the User object.

There is a ternary relation between the AuctionHouse object, the Seller object and the Product object. This relation reflects the real world, as the submission of a product concerns the seller, the auction house and the product. This relation has as attributes the start-up value and a deadline date, which is by when the seller's product must be auctioned.

The other ternary relation is that between the Buyer object, the Auction object and the AuctionHouse object. Its attributes are the value and the time of a buyer's bid.

Between the Auction object and the Product object, there is a 1-1 relation. That is because each product is auctioned by one and only one auction, and each auction is created for one product.

LOGICAL DESIGN

This section describes the objects of the system and how they relate and co-operate with each other in detail.

Description of Applets

AuctionApplet A user can participate in an auction, by invoking this applet. He/she can choose an auction, see some information relative to this auction (like product description, start value, and the number of bidders), and submit a bid.

ObserveApplet A user can observe an auction, by invoking this applet, even if he/she has not registered before. The observed auction can be chosen from a list of auctions, but bidding is not allowed.

TransactionApplet By invoking this applet, a user can participate in an auction or declare a product, giving the appropriate login and password. This is the authentication applet.

DeclarationApplet By invoking this applet a user can declare the product that he/she wants to submit for an auction. A description of the product is given along with the desired price and a deadline, by which the product must be auctioned.

RegistrySellerApplet/RegistryBuyerApplet These applets allow registration to the system.

AdminApplet By this applet the administrator of the system can control the whole system from any computer. The administrator has the capability to check the registry file and to remove users. Also, he can see the already auctioned products, the successful bidders and the associated history file (how users were bidding). He/she can set a Standby auction Inactive, and then set it as Standby[*] . Furthermore he can check the state of all auctions in the system, and if he wants he can change it. The lifecycle of an auction starts when a user submits it to the system, and ends when a buyer obtains the product through the auction.

Description of Objects

AuctionHouse This is the main object of the system. It takes care of the creation, initialization and the synchronization of the other objects.

As the system begins, it checks if there is any product to be auctioned. If there are some, it creates an auction for each product. The users' authentication take place with the TransactionApplet which interacts with the object AuctionHouse. If a product declaration is available, AuctionHouse saves it in the appropriate database, and creates an auction for it. If an auction has completed, AuctionHouse kills the associated object to free resources from the system.

Auction This object auctions a product and it is created by the object AuctionHouse, if there is a product to be auctioned.

Besides its core function as implied by the description, it informs the users of that auction about the current value of the product, the number of bidders and the current hit number of the auction (an auction is concluded, as usual, on three successive hits).

Buyer This object's role is to serve a user's requests for registration, then giving him/her the capability to participate in an auction.

It checks a login and a password if they exist in the database, and creates new entries. This object is a kind of cache. It receives the user's request, and then it forwards it to the object HashBuyer, which manages the database.

Product This object serves the sellers' request for a product to be auctioned.

[*] An Auction may be *Active* (a user has already made a bid), or *Standby*, (no user has made a bid yet), or *Inactive* (the auction is not opened to users yet).

It announces to AuctionHouse a new product. This object is a kind of cache too. It receives the requests from the DeclarationApplet for a product declaration and forwards them to the AuctionHouse.

SBuyer This object stores information about buyers.

SSeller This object stores information about sellers.

Infoproduct This object stores information about the product that will be auctioned.

Database management

Voyager 2.0.0 (beta 2) provides programmers with a light database. In this database a programmer can only store Voyager's objects of the same type. The above feature leads us to use more than one Voyager servers, one for each database we need. It should be noted that this is a limitation of the current implementation of the Voyager platform.

SoldProduct/VectorProduct The main job of these objects is to store objects of type Infoproduct. The object SoldProduct stores InfoProduct as an already sold product. Object VectorProduct stores InfoProduct as a product that will be auctioned.

HashBuyer/HashSeller These two objects store Sbuyer and Sseller objects equivalent into hash tables.

Both objects manage the hash tables via standard Java methods.

Interaction of Objects

In this chapter we describe how the objects interact with each other and the messages that they exchange.

Transactions If a user wants to declare a product or to participate in an auction, he/she must give his login and password. These will be checked by the system and if they are valid, the user can declare a product or participate in an auction. The steps that the system follows is (shown in Figure 2):

1.If the user wants to enter in the system as a seller, TransactionApplet will send a message to object Seller with the user's login and password as content, to check them.

2.Object Seller communicates with object HashSellers which manages the database of sellers.

3.If the entry exists, the object HashSeller will return the appropriate value to the object Seller.

4.Object Seller forwards the received value to TransactionApplet.

5.If TransactionApplet receives the message that the user is valid, it will inform object AuctionHouse about the fact that a user wants to enter in the system. Then it sends to AuctionHouse the user's IP address, login and password. The TransactionApplet then connects the user with DeclarationApplet from which he/she can declare a product.

The same procedure will be followed in the case that the user is a buyer. The only difference is in step 5, where now the TransactionApplet will connect the user to AuctionApplet instead of DeclarationApplet (not shown in the figure below).

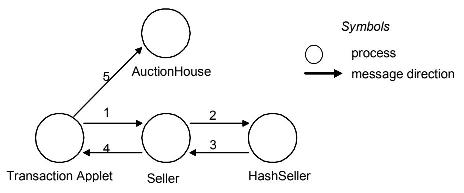

Figure 2: Authentication of users.

User Registry The procedure for registering a user is the following (see Figure 3):

1.After the user has entered his/her details, the RegistryApplet sends a message to object Seller to check if there is an entry with the same login and password.

2.Object Seller communicates with object HashSeller which manages the appropriate database.

3.Object HashSeller checks the login and password and returns to object Seller the appropriate message.

4.Object Seller forwards the received message to RegistrySellerApplet. Then the user is informed about the progress of the registration.

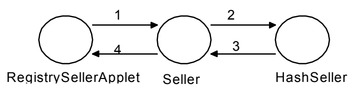

Figure 3: User registration.

Observe an Auction A user can observe an auction without having registered before. This can be done through the ObserveApplet. The procedure is the following (see Figure 4):

1.The object AuctionHouse is informed that a user wants to enter in the system to observe an auction.

2.The object AuctionHouse informs the user (through ObserveApplet) about the schedule of the auctions, these are the auctions that take place right at that time.

3.The user decides which auction to observe and connects with it.

4.The object Auction informs ObserveApplet for the current state of the auction.

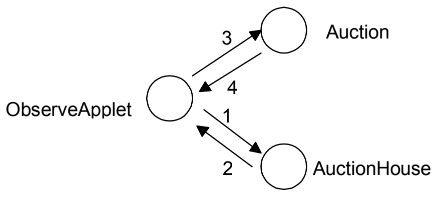

Figure 4: Auction observation.

Initialization At program startup, the object AuctionHouse creates all the other objects. If the system has been shut down before, objects which manage the databases are woken up due to their persistence, and the object AuctionHouse is connected to them.

If there are any products that have not been auctioned yet, the object AuctionHouse will create an auction for each of them. After that, the object AuctionHouse awaits some event. This event will be generated by a user, or by the system administrator or by an object of the system (auction termination for example).

Product Declaration If a user wants to declare a product to be auctioned, the following procedure is carried out (see Figure 5):

1.The DeclarationApplet is connected to object Product and sends it a product description.

2.Object Product communicates with AuctionHouse and informs it about the new product which must be auctioned.

3.AuctionHouse creates an auction for this product and update its schedule about the new auctions.

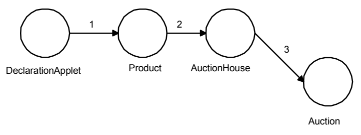

Figure 5: Product declaration.

Carrying out an Auction If a user wants to participate in an auction, he/she can do that through the AuctionApplet (see Figure 6).

1.The AuctionApplet is connected with object AuctionHouse and sends it the user's IP address.

2.The object AuctionHouse informs the user, through the AuctionApplet, about the auction schedule.

3.After the user selects the desired auction, AuctionApplet will be connected with that auction and will send it the user's IP address.

4.The object Auction sends to AuctionApplet information about the current state of the auction, like the startup value, the current value and the number of occupants at that time.

5.After that, the user can participate in the auction. Note that each auction creates an auxiliary object which helps the Auction object to keep track of hit counts.

6.When the auction is finished, the object Auction announce to users the successful bidder, and informs the AuctionHouse about the progress of the auction.

7.The AuctionHouse object kills the Auction object, to free resources from the system, and stores the history file of that auction.

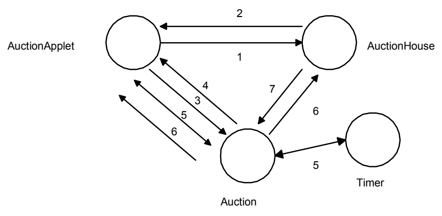

Figure 6: Carrying out an auction.

System Administration The system administrator can control the whole system through the AdminApplet. This applet interacts with the main objects of the system (AuctionHouse, SoldProduct, HashBuyer and HashSeller).

When AdminApplet is started (see Figure 7), it is informed by the AuctionHouse about the schedule of the auctions and the state of the auctions. By the SoldProduct, it obtains information, about the list of products already being auctioned. By the HashBuyer and HashSeller it obtains information about the list of buyers and sellers that have already been registered in the system.

The system administrator can interact with any of the above objects to perform administrative operations. These operations may be the removal of a user (seller or buyer) from the system, the changing of an auction's state (from Active to Standby and vice-versa), the removal of an already auctioned product from the appropriate database, and finally the display of a history file of an auction and of information about the winner of that auction.

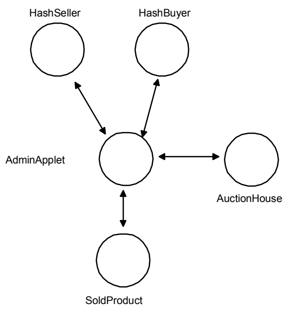

Figure 7: System administration.

Voyager Features

We will describe below the basic concepts of Voyager, which are used for the development of the system.

Interfaces, classes and objects Interfaces are Java constructs. An interface doesn't contain code and data; it defines a set of methods, which will have to be provided by every class, that implements the interface. Interfaces provide a flexible way to implement multiple inheritance mechanisms without all the overhead incurred by multiple inheritance per se.

Voyager-enabled programs When a Voyager-enabled program starts, it automatically spawns threads that provide timing services, perform distributed garbage collection and accept network traffic. Each Voyager-enabled program has a network address consisting of its host name and a communication port number, which is an integer unique to the host. Port numbers are usually randomly allocated to programs. This is sufficient for clients communicating with remote objects. However, if a program will be addressed by other programs, somebody can assign a known port number to the program at start-up.

Constructing a remote object and a proxy (virtual reference) to it A remote object is created with a simple instruction including the address of the destination program, where we want to create the remote object. If the class code of the remote object doesn't exist in the destination program, Voyager's network class loader automatically loads class code in it.

When a remote object is constructed, a proxy object whose class implements the same interfaces as the remote object is returned. The proxy can receive messages, forward them to the object, receive the return value, and pass the return value on to the original sender.

Value-adding Interfaces When a remote object is constructed, Voyager wraps the object in a *Voyager component*. A Voyager component extends the object with special interfaces (IIdentity, IMobility, ILifecycle, IProperty,and IProxy) that add value to it.

Lifecycle and Persistence A programmer can specify and change the life-cycle of an object. The life-cycle is the duration of existence for an object. When an object reaches the end of its life-cycle, it dies and is garbage collected. The default life-cycle for an object is to live as long as at least one regular Java reference is made or a proxy points to it.

Remote objects can also be persistent. Persistence is achieved by writing the object to a database, which is specified at Voyager start-up. Later, when a message is delivered to a persistent object not currently in memory, Voyager loads the object from the database and activates it.

Mobility and Moving Objects By default, all serializable Voyager objects are mobile. Mobility is the capability of objects to be moved from one virtual machine to another within a computer or a network. At the same time, garbage collection mechanisms are fully supported with persistent and mobile objects.

EPILOGUE - TOWARDS A WORKABLE SYSTEM

The above system has been developed to the level of a research prototype (Boug98) and is the basis of a commercial product under development, to serve the varying needs of the supplier community on auction variant services. The system has not been yet tested in a real-life situation (within a commercial context) due to rather slow pace of adoption of the WWW as a commercial outlet. Obviously, this is a consequence of the relative immaturity of the home market and the lack of readiness to accept this kind of solutions. The new product under development is based on the concept of stimulating product demand by on-line ordering and subsequent price reductions; it is hoped that several companies will install a version by late 1998 - early 1999. Having said that, the system has been extensively tested in a laboratory situation, where besides a rather slow start (an initialization overhead incurred by the auction house only), the registration, product declaration and bidding stages have been demonstrated to be efficient.

The seamless integration of Java and Voyager over the WWW has meant that a truly distributed system has been developed with the potential to easily adopt it to other real time electronic commerce applications. Among our research and development priorities are the full appreciation of the bandwidth limitations under real-life situations, the maturing of the existing functionality and the development of autonomous bidding agents with the help of machine learning techniques for behaviour modelling. It is our belief that the current system provides a suitable and indispensable infrastructure for developing solutions to such problems.

REFERENCES

Bouganis C. and D. Koukopoulos. 1998. *An Electronic Auction System over the WWW*. Diploma Thesis at the Dept. of Computer Engineering and Informatics, University of Patras, Greece (in Greek).

ABOUT THE AUTHORS

Christos Bouganis was born in Athens, Greece, on November 14, 1975. He received his diploma from the department of Computer Engineering and Informatics of the University of Patras in 1998. His diploma thesis addressed the research and development of an electronic auction system over the WWW. As of October 1998, he is studying towards an M.Sc. in Telecommunications and Signal Processing, in Imperial College, UK.

Dimitrios Koukopoulos was born in Karditsa, Greece, on July 7, 1975. He received his diploma from the department of Computer Engineering and Informatics of the University of Patras in 1998. His diploma thesis addressed the research and development of an electronic auction system over the WWW. Based on that work, he worked at the Computer Technology Institute towards the development of a product, commissioned by a WWW services company. As of October 1998, he is studying towards an M.Sc. in Telecommunications and Signal Processing, in Imperial College, UK.

Dimitrios Kalles was born in Volos, Greece, on March 18, 1969. He received his diploma from the department of Computer Engineering and Informatics of the University of Patras in 1991, then moved on to UMIST (Manchester, UK) to obtain an M.Sc. (1993) and a Ph.D. (1995) from the Department of Computation. He joined the Computer Technology Institute in 1997 and now works as a researcher supervising applied research projects and the development of small scale basic research prototypes. His main interests are in the areas of machine learning, geographical information systems and image processing.