The following paper was originally published in the
Proceedings of the Second USENIX Workshop on Electronic Commerce
Oakland, California, November 1996

Information and Interaction in MarketSpace—
Towards an Open Agent-based Market Infrastructure

Joakim Eriksson, Niclas Finne, Sverker Janson
Swedish Institute of Computer Science

Information and interaction in MarketSpace—towards an open agent-based market infrastructure

Joakim Eriksson Niclas Finne Sverker Janson

Swedish Institute of Computer Science
SICS, Box 1263, S-164 28 Kista, Sweden
{joakime,nfi,sverker}@sics.se

Abstract

The lack of structure of information and interaction in current web-based electronic commerce makes partial or full automation infeasible. We describe the first steps towards an open agent-based market infrastructure, with well-defined information and interaction models allowing agents to locate relevant market participants, exchange interests, and negotiate deals. We also describe an architecture and a prototype based on these ideas, and illustrate the operation of the prototype in an example scenario.

1 Introduction

Current web-based commerce is by and large a replica of real-world commerce. Customers are supposed to visit various electronic storefronts, inspect the digital displays of goods and services, and, perhaps after some comparison shopping, place an order.

Unfortunately they have to do it in person. Attempts at supporting the customer through "intelligent agents" are guaranteed to be in vain, unless the underlying model is changed. It would have to be one bright agent to, using the web as is, find a few good deals on a fridge at nearby resellers, haggle prices and delivery conditions, and then present the three best alternatives. Which minute fraction of the 50000 hits on "refrigerator" in AltaVista is at all relevant? In these, what part of the text is the price, the model, the delivery conditions?

Needless to say, much better models are possible, in which the above would be a very basic service, and one such model will sooner or later supplant the web-based models. The web, and associated technologies such as Java, will remain as the user interface, but a new infrastructure will emerge that is targeted to the task of creating a near-perfect global market.

In this paper, we present work towards such an infrastructure, characterized by *openness* and by being based on a paradigm of *interacting agents*.

We would like the market to be open like the web. Nobody should own it. Anyone should be able to enter it. Anyone should be able to offer any kind of service, such as brokering. This is typically not the case with current markets, where all information is owned by the market operator and not available for use by others [1, 2].

A market has to support more activities than just finding the desired product or customer. Well-defined interaction protocols are needed for negotiation. Ideally, these protocols should make sense for human-human, human-agent and agent-agent interaction alike, to make possible any mix of human and automated participants in the market. (For agent-based approaches, see, e.g., Kasbah [3] and MAGMA [4]. For pertaining discussions see [5].)

Note that although the issues of security and payment are strongly emphasized in most works on electronic commerce, they are quite orthogonal here. Any future standards will do.

Organization of this document

In the remaining sections of this paper, we outline the information and interaction models that support our preliminary MarketSpace infrastructure, describe the design of a MarketSpace server architecture and prototype, and finally offer a few concluding remarks.

2 Information and interaction

We will need an information model in which we can describe information about participants and their interests, different products, delivery specification, etc. The model must be flexible in a way that a new product can be described and used, and structured so that the information described is easy to parse for a program. There must also be an interaction model, setting up rules for how the participants in MarketSpace can negotiate with each other.

In the rest of this section we will introduce our approach to the information and interaction models for MarketSpace.

Information

Interests are the essential pieces of information in a market. They describe what a participant (person, company, organization, etc) is concerned about, curious about, intends to do or wants. There are several different kind of interests from simple forms of interests like "I am interested in cats" to more complicated like "I want to buy all things needed to build a spacecraft and hire personnel to build it". Since the goal of a market participant is to make deals, an interest can be said to describe a set of potential deals. We will now try to make these concepts a little more precise.

What we will need to know about a *deal* is that it has a type, some participants involved, and a time when it is entered. As basic types we have (the names of) *participants* in the market, which are either human or programmed, and *time points*. The *deal type* includes everything else, such as what is sold, delivery conditions, etc. With each deal type is associated a set of *rôles*, which are the parts played by participants in a deal, such as buyer or seller. A deal assigns participants to all the rôles in its type.

An *interest* is a set of deals. An *expression of interest (eoi)* is a representation of an interest.

The interaction model

Based on the interests, we define a very simple speech-act style interaction protocol. It consists of *information messages*: ask and tell, and *negotiation messages*: propose, reply, accept, and reject.

ask(A, B, eoi, id) A asks B for information on interests, giving an eoi (possibly but not necessarily its own interests) as guidance. Replies can be given by tell.

tell(A, B, eoi), tell(A, B, eoi, id) A tells B about some interests (possibly its own). If this is intended as an answer, the question can be referred to by its id.

propose(A, B, eoi, id) A initiates negotiation with B, giving an interest as guidance. Replies are given by reply, accept, or reject.

reply(A, B, id, eoi, id) A replies to B's latest bid. The interest given need not have any special relation to the bid. (It is up to the participants involved to assess the progress of the negotiation and interrupt it when necessary.)

accept(A, B, id, id) A accepts B's latest bid. This is equivalent to signing a contract. The reply is to accept or reject.

reject(A, B, id) A aborts the current negotiation with B.

We are currently exploring conversational styles and idioms using these basic messages, rather than integrating capabilities through complex message types. This will allow some, simpler, participants to have a more naive view of interaction, while more advanced participants can recognize and enjoy the benefits of more elaborate patterns of interaction.

Other agent communication languages include KQML (see, e.g., [6]), which, being focussed on the communication of knowledge, does not offer specific performatives for negotiation.

3 A MarketSpace server

We have developed a MarketSpace server architecture and prototype based on the ideas presented above. The prototype was developed using SICStus Prolog and Prolog Objects [7].

The rôle of this server in the MarketSpace infrastructure is both to support human participants, with whom it will interact by emulating a web server, and programmed participants (agents), for which it will serve as a scheduler and communications mediator.

The market server architecture

The MarketSpace server architecture has three main components (see Figure 4):

- the *kernel*, which handles the events that schedule activities in the system and communication with the outside world;
- the *protocol handlers*, which register the protocols they listen to and get the corresponding events from the event handler when data arrives;
- the *agent environment*, which implements the runtime environment for agents.

Two event types are built-in: The *time* events are used by system components (objects) that wish to

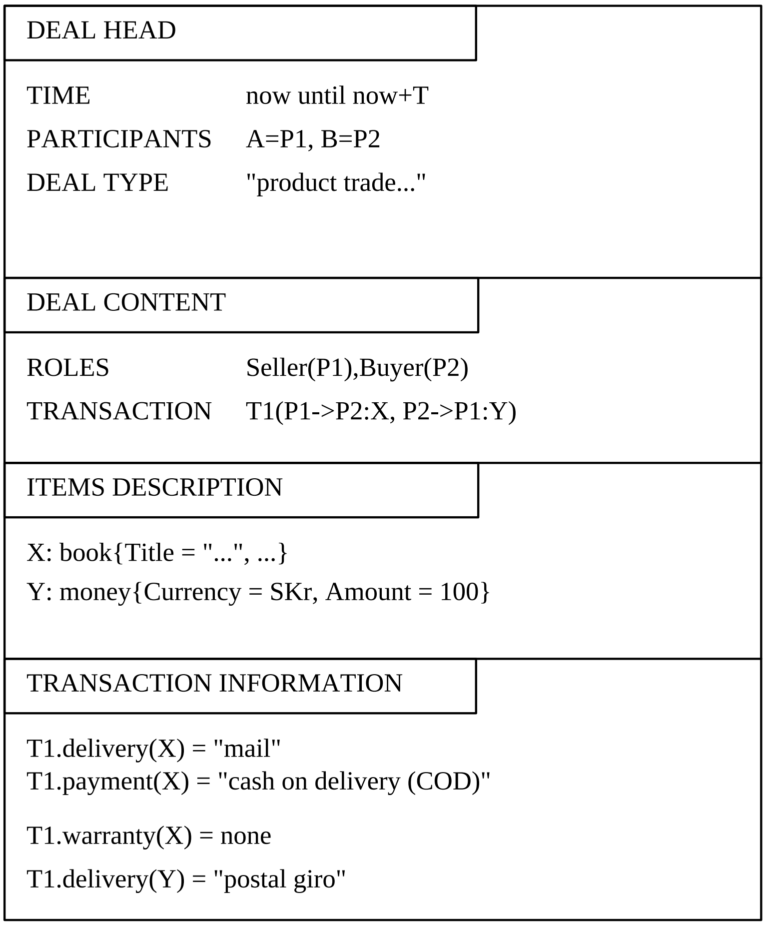

Figure 1: Example of a possible EOI

be scheduled at regular time intervals and the *stream* events signal the arrival of data from the outside world to system components such as the protocol handlers.

The object model

The implementation relies heavily on the Prolog Objects extension of SICStus Prolog. The basic objects (prototypes) are *root*, which adds initialization and finalization capabilities, and *persistent*, which also adds the ability to save (checkpoint) and recreate state from secondary storage. The persistence mechanism is also used for migration of objects.

The kernel

The kernel is the main engine of the system and has two important rôles: one is to generate stream and time events, the other is to distribute events among subscribing objects. The kernel consists of the following components:

- the *stream handler* object, which handles the communication from the outside world with the system using socket and stream communication;
- the *time handler* object, which generates time and delay events;
- the *event handler*, which is the main kernel object, and handles scheduling of events to the system objects.

The kernel also gives higher level components the possibility to: subscribe for events, add events, and register new event types and protocols.

The protocol handlers

The protocol handlers handle data arriving from the outside world. They register the protocols and subscribes to stream events from the event handler.

The *HTTP handler* provides World Wide Web server capabilities and forwards parsed HTTP messages to other objects. (A Prolog DCG parser could

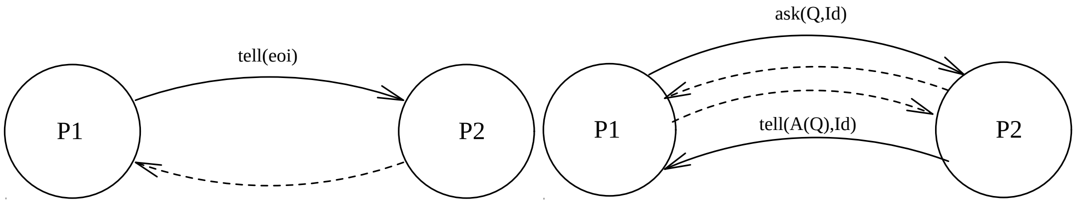

Figure 2: Tell and Ask

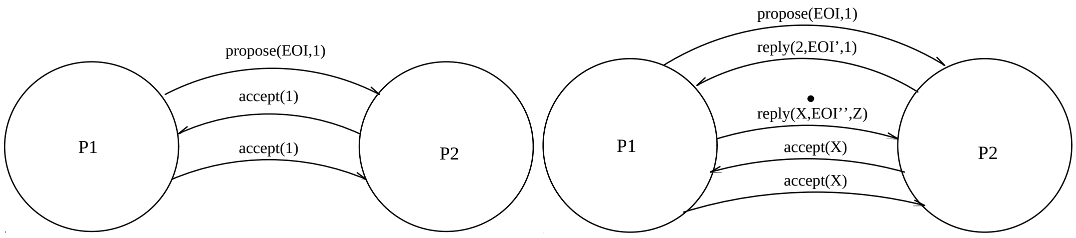

Figure 3: Negotiation

easily be derived from the HTTP specification [8].) Objects can subscribe to a specific path whereby web accesses matching that path will be forwarded to the subscriber. The following system components subscribe to HTTP events:

- the *html server*, which subscribes to the empty path and receives all web requests not caught by other subscribers, replying with files matching the web request;
- the *cgi-bin server*, which subscribes to the path "`cgi-bin`" and handles user registration and authentication;
- the *cgi object*, which subscribes to the path "`cgi-object`" and provides a way to inspect any runtime objects from the web.

The *DOP handler* handles communication with the DOP, Distributed Object Protocol, which is a simple protocol (defined by us) for distributing Prolog Objects.

The *TERM handler* handles communication with the TERM protocol. TERM messages are parsed into code and executed within the server context. This makes it possible to examine and control a running server which is very useful for debugging.

The agent environment

At the top level of the architecture we have built a simple agent environment in which programmed participants in MarketSpace can reside. The agent environment implements a set of features useful for agents. These are:

- registration of requirements and description;
- agent communication;
- retrieval of information about other agents.

Although this environment offers some of the services needed by agents, it has to be extended. For example, there will be a need to manage ontologies for the content language(s) of agent communications.

Web-based users register in the server and are give a participant name in MarketSpace in the same name space, and indistinguishable from, the names of agents. Agents and humans need not, and should not, know if they are talking to humans or agents.

Interests and interactions

For the present, the interests have a simple representation.

- Items, like products to sell, are represented as objects with a set of methods for accessing information about concepts and instance values.
- Deals are represented as different objects according to deal type.
- EOIs, Expressions Of Interest, are represented as objects with a list of deal objects. Methods for accessing EOI information (like deals) exists as well as set operations like *union*, *intersect* and *partition*

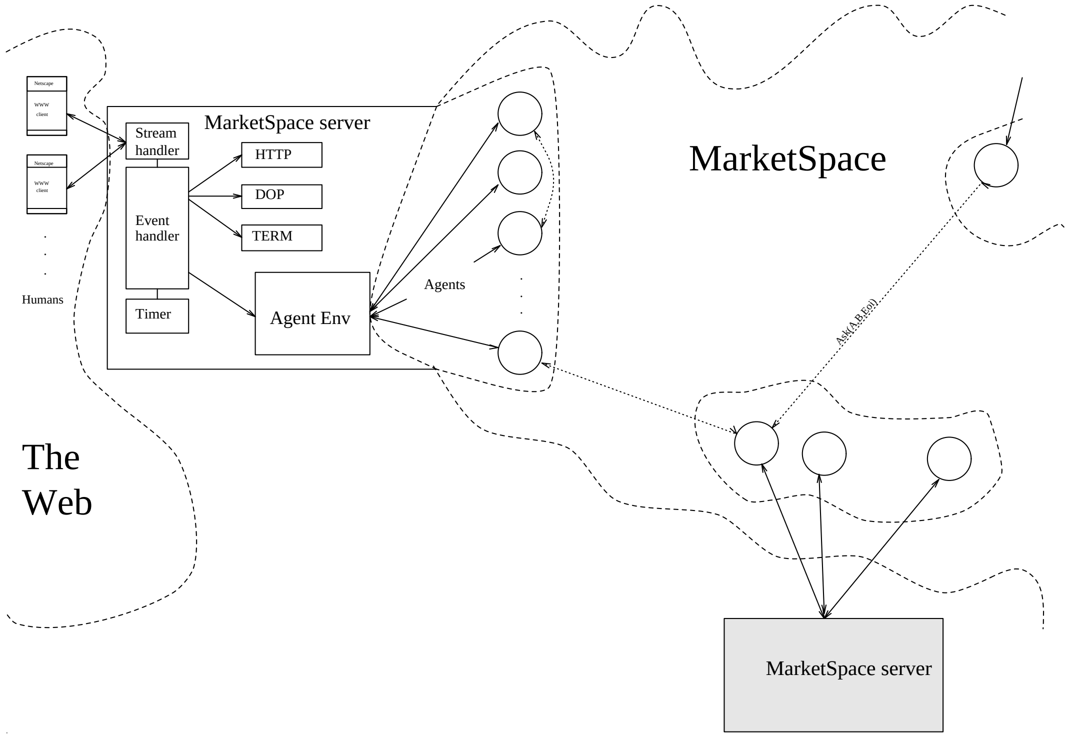

Figure 4: The basic MarketSpace architecture

The interaction model is implemented using interaction (or message) objects. They inherit from the persistent object (enabling migration) and have methods for accessing message information and sending messages to other agents.

Example

Finally, we illustrate the use of this architecture and system by an interaction between two servers: the *blue server* and the *red server*.

The blue server provides a web interface to interact with human participants, who can create, change and remove interests from their interest store, and engage in negotiations, all through dynamic web pages. The blue server hosts a collector agent, which looks for new interests and forwards them to a broker agent at the red server.

The red server lets human participants specify their interests as URLs to web pages in which the interest is given in a special syntax. The red server fetches the specified web page and parses the interest. The red server also hosts a broker agent to which a human participant can delegate the task of negotiating a deal, giving upper and lower price limits.

In addition, a *log server* subscribes to log events from the other servers, and displays what is going on in MarketSpace.

Example scenario

The following simple scenario illustrates a possible interaction in the marketspace.

- User A on the blue server specifies that she wants to buy a book. The user does this by creating an interest and specify the rôle to act (buyer, seller) and the product to buy or sell. This interest is stored in a user store which is accessible by all agents in the marketspace.
- User B on the red server specifies that she wants to sell a book by specifying a URL to a web page (using the special format for interests). The web page is fetched and information about the interest is extracted and put in the user store.
- User B on the red server delegates her interest to a broker agent through an agent interface to which the broker agent has registered. When delegating the interest, the user specifies an acceptable price and a suitable starting price. During negotiation, the broker agent starts with the starting price and slowly goes to the acceptable price trying to get the best price.
- A collector agent in the blue server collects all local interests and sends them to the broker agent. At the moment this is done at regular

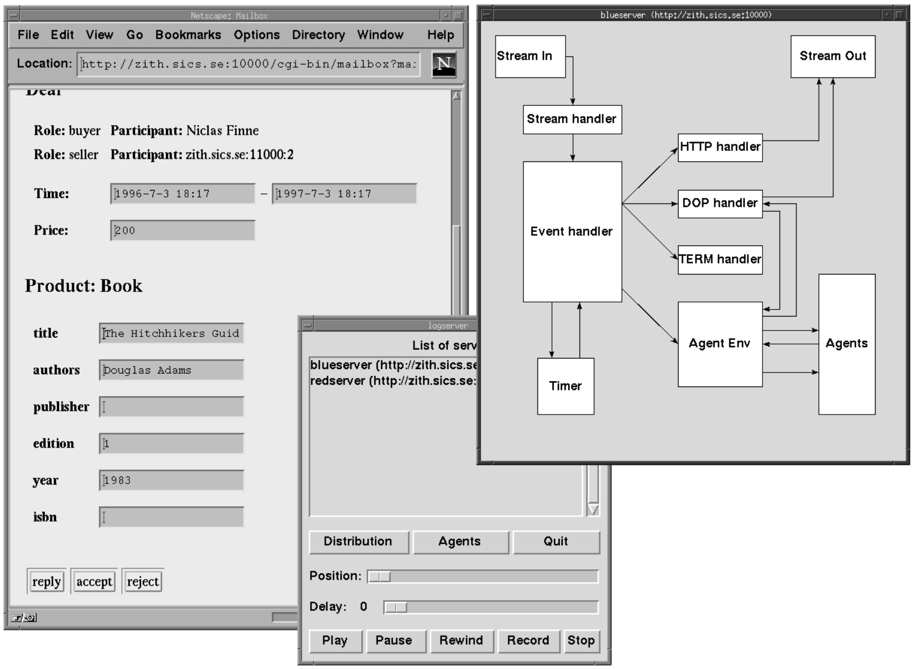

Figure 5: Snapshot of a proposal given to a user (left window) and the log handler

intervals, but in the future it will probably be some form of subscription. To collect all local interests, the collector agent starts by asking the administration agent for all local agents (which also includes users). Then the agent asks each such agent for their interests and when the agent happens to be a user, the user handler catches the message and answers it by fetching the users interests from the user store. The union of all interests found are sent to the broker agent at the red server.

- The broker agent receives the interest and checks for a matching interest by taking the intersection between the received interest and the delegated interest. If the intersection not is empty, the broker agent changes some information as for example price and sends a proposal to the agent that owns the interest. In this example it is user A at the blue server but it could just as well been a programmed participant.

- User A receives a message from the broker (see Figure 5) and the message is converted into a web page when the user reads it. The web page describes who the sender is, the type of message and its contents together with buttons to do various actions with it. For example, the user can choose between *accept*, *reject* and *reply* when replying to a proposal. The user chooses to negotiate by changing the price and selecting *reply*. A reply message is created and sent to the broker agent.

- The user and the broker agent negotiates until both accepts a deal (or someone rejects in which case the broker agent tries to find another agent to negotiate with).

- When the broker agent finally settles a deal, the user B is informed.

4 Conclusions and future work

We have defined a first version of the MarketSpace architecture for open agent-based market infrastructures, allowing human and programmed participants to co-exist smoothly. A first prototype has demonstrated the feasibility of the approach. This work is the starting point for a range of activities. In the short term future we will:

- investigate representation languages and ontologies for interests, involving issues such as product/service categories and transaction types;

- explore the integration of emerging payment mechanisms;

- complement the present platform with distributed naming and directory services, possibly based on KQML-like knowledge sharing;

- explore the possibility to support MarketSpace participants by "defection detection" and protocols for "social control" [9];
- develop a range of MarketSpace components to support various rôles, in particular professional vendors and suppliers of services, brokers and other mediation services, and interfaces to other market places, in particular services on the web, publishing their information and in effect creating a unified market.

Finally, for those who are interested in exploring systems of this kind using Prolog, we offer our present prototype as a starting point (see `http://www.sics.se/isl/commerce/`, where also other papers and more information on the project can be found).

References

[1] Internet Shopping Network. URL: http://www.internet.net.

[2] Polycon AB. The webra marketplace. URL: http://www.polycon.fi/webra.

[3] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of PAAM'96*, 1996.

[4] Maksim B. Tsvetovatyy and Maria Gini. Toward a virtual marketplace: Architectures and strategies. In *Proceedings of PAAM96*, 1996.

[5] Jim R. Oliver. On automated negotiation and electronic commerce. In *Proceedings of 29th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, 1996.

[6] Tim Finin and Richard Fritzson. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, 1994.

[7] Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, 1995. Also available via http://www.sics.se/isl/sicstus.html.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0. Available at http://www.w3.org/pub/WWW-/Protocols/HTTP1.0/draft-ietf-http-spec.html.

[9] L. Rasmusson, A. Rasmusson, and S. Janson. Reactive security and social control. In *Proceedings of 19th National Information Systems Security Conference*. NCSC, 1996.