A Trader for Services in a Scientific Literature Market [1]

Michael Christoffel

Universität Karlsruhe, Fakultät für Informatik
D 76128 Karlsruhe, Germany
christof@ira.uka.de

Abstract. The success of the Internet has made some dramatic changes in the way scientists and students are supplied with scientific literature. They can now operate in an information market, where the value of a piece of information is determined by the law of supply and demand. Traders assist the customer in this market. This paper describes a basic approach to trading in the open, distributed, and heterogeneous environment of a market of scientific information.

1 Introduction

The success of the Internet, especially the World Wide Web, has done away with the monopoly of the local library and the local bookstore as the suppliers of information. Publishers, electronic bookstores, database services are as easy accessible. They all operate in a highly competitive environment, a market where they vie for the attraction of the customers.

This paper considers the market of scientific literature, with university people as customers. The information they deal is books, journals, conference proceedings, technical reports, and bibliographic data. The services they utilize come from libraries, publishing houses, technical report servers, document delivery services, and electronic bookstores. To locate among them the providers that deliver the desired piece of information under prescribed quality conditions for a fair price, the customer needs assistance by the system. System components that offer such assistance are referred to as *traders*. The objective of this paper is to outline the characteristics of a trader for the market of scientific literature.

2 Underlying Environment

The trader described in this paper is part of the UniCats project (a UNiversal Integration of Catalogues based on an Agent-supported Trading and wrapping System) at the University of Karlsruhe [6].

There are two main system components beside the trader:

A *user agent* establishes the system connection towards the customer. It transforms the customer's demand into a query to the trader, as well as follow-up queries to the providers in order to acquire bibliographic data or complete documents. It displays the collected results to the customers.

[1] This project is supported by the Deutsche Forschungsgemeinschaft (DFG).

A *wrapper* establishes communication with the providers using their HTML-interface. It transforms the queries sent by user agents or traders and returns the results delivered from the sources. Moreover, it creates metadata descriptions of the providers which are passed on to the traders. Wrappers are tailored to the individual wrapped information source with the help of a wrapper generator.

In terms of the I^3 reference architecture [8] user agents correspond to mediators, traders are a special kind of facilitator, wrappers have an identical function.

While it is obvious in an open system to have a dynamically changing number of customers and providers, we also assume that our environment may include several – probably competitive – traders. Further, information sources should liberally be attached to and detached from the network and thus be only loosely coupled to the system. Consequently, a trader has no access to the provider's database other than the HTML-interface the provider has installed.

The UniCats project has an underlying agent framework to cover the heterogeneity and to support communications between system components on the basis of a sufficiently high-level data model to express the semantics of the exchanged information. The agents, which may run on any platform, communicate between each other by use of TCP/IP connections transmitting XML documents. The framework offers two different kinds of communication modes:

- *Group connections* enable message broadcast between all agents belonging to a group. For instance, the members of a trader federation can share messages this way. A resource manager is used to filter those messages out of the message stream that may be interesting for the agent.
- *Direct connections* between two arbitrary agents are used for secure electronic commerce transactions. To open a direct connection, an agent must have an individual socket. The transmitted objects are extracted out of the XML character string and transformed into DOM objects (document object model) which may be interpreted with the help of a special DTD (Document Type Definition). This process may be influenced by the agent. Data exchange (including document exchange), billing and payment will be transacted with a DTD for the OTP (open trading protocol).

By using XML, the framework remains independent of the used programming language (we use Java as implementation language).

3 Attribute Assignments and Profiles

For trading purposes, we need descriptions of service providers and offered services that abstracts from the actual appearance of the source. We call this description the *metadata* of the provider. The metadata are used to define both existing providers and the customer's intentions.

Each property of a service provider may be expressed by an attribute A, and each attribute must have an attribute type T_A. For example, we can express delivery time and way of delivery by the following attributes:

attribute delivery_time: Cardinal;
attribute delivery_way: (online, email, fax, postal, pick up)

For each attribute A, we assign one or more values (*attribute values*) of the type T_A, and attach to each attribute value a weight from the interval [0,1]; omitting this factor means a weight of 1. An attribute together with its assigned values and weights is called an *attribute assignment*.

Usually, attributes give descriptions of a source, such as the kind of provider (library, document service etc.), the covered scientific areas, or the estimated cost of an access to a source. They are also used to define restrictions for the work of the trader, e.g., maximum charge.

A value of an attribute may depend on the value of another attribute. For instance, the delivery time may depend on the delivery method. It is obvious that a trader can work the better the more dependencies the attribute set contains. But it will be much more complicated to acquire values for the attributes if these values change with the values of other attributes. Moreover, query handling will become more difficult.

The set of all attribute assignments for an information source is called the *profile* of this source. Some attributes can be assigned only one attribute at a time and are called *univalent* as opposed to *multivalent* attributes. Consider the following example:

```
topic := mathematics, physics[0.5], chemistry[0.4];
delivery_way := email, fax;
delivery_time := 1 if delivery_way = email,
                 12 if delivery_way = fax;
```

This source is specialized on mathematical literature but also provides, to a lesser degree as predicted by the weight, documents concerning physics or chemistry. It delivers by fax or email within different time frames. Topic and delivery way are multivalent, delivery time is univalent and dependent.

The profile of a service provider will be handed over to the trader when the provider registers with the trader. This can be done by a person with good knowledge of the source, such as a librarian, with the help of a questionnaire created by the trader. Of course, the profile can be updated at any time.

Another possibility to determine attribute values is the use of experience. Experience is gained either in an experimental way by sending test queries to the provider (remember that the trader has no direct access to the underlying data source) or by obtaining evaluations and ratings from the user agent.

Test queries will be the best way to get facts such as the average response time of a source. However, the average time a customer spent with research on a source can only be determined after this research is done. The same holds for the number of interesting documents found in a source or the actual cost of the session. Clearly, experiences add a learning capability to the trader.

4 Query Handling

4.1 User Queries

We define the customer's demand in the same way we describe a provider: A user query consists of a set of attribute assignments. However, univalent attributes in a user query may have several values in an attribute assignment:

```
delivery_time := 1 [1.0], 2 [0.8], 4 [0.2];
```

The customer in this example prefers a delivery time of one hour, also accepts a delivery time of two hours, but does not want to want to wait longer than four hours. Also, dependencies are not considered in a user query.

The following user query expresses that a user wants to have online documents about mathematics or computer science written in English or French (but preferable in English):

delivery_way := online;
topic := mathematics, computer_science;
language := English, French [0.5];

4.2 Neighborhood Relations

Consider the following user query:

delivery_time := 2;
topic := zoology;

Certainly, a source that provides literature about biology online will fulfil the user's intention. However, the source will escape the attention of a trader that does not know that a delivery time of less than two hours is quite as good (or better) than the specified value, or that biology includes zoology. These properties cannot be expressed by the attribute type. We have to add a formalization of general knowledge about the attributes and their values.

For an attribute A of the attribute set, we call each mapping $T_A,[0,1]\rightarrow T_A$ *neighborhood relation* of A. We use the following notation:

$$A: a_1 \xrightarrow{w} a_2 \qquad a_1, a_2 \in T_A, w \in [0,1]$$

A neighborhood relation assigns additional attribute values to an attribute value and attach a weighting factor, as shown in the following example:

$$\text{zoology} \xrightarrow{1.0} \text{biology}$$
$$2 \xrightarrow{1.0} 0$$
$$2 \xrightarrow{0.9} 1$$
$$2 \xrightarrow{0.8} 2$$

Using these relations, the above user query will be translated into a query that comes closer to the user's intention:

delivery_time := 0 [1.0], 1 [0.9], 2 [0.8];
topic := zoology [1.0], biology [1.0];

In practice, neighborhood relations will be declared in a more general way, such as:

$$\text{delivery\_time}: \quad a \xrightarrow{1.0} x \quad \text{for } 0 \le x < \tfrac{a}{2}$$

We denote the transformation of a user query Q by trans(Q). The transformed user queries are the starting point for the query handling which is introduced in the following section.

4.3 Trading Algorithm

Attributes define the dimensions of a multidimensional *attribute space*. Each profile and each user query determine a point cloud in this space. Consequently, matching a profile to a given user query can be done by computing some measure of conformity between point clouds. The process can coarsely be illustrated with the following algorithm:

```
input    Q,              // user query
         min_results,    // requested minimal and
         max_results     // maximal number of results
repeat
  results := ∅
  for each profile P do
     c := conf (trans (Q), P)
     if c > 0 then
        find position k in results with
           ∀i<k [conf (trans (Q), results[i])>c]
           ∧ ∀i≥k [conf (trans (Q), results[i])≤c]
        insert P to results after position k
     end
  end
  if #results > max_results then
     delete results after position max_results
  elsif #results < min_results and Q ≠ ∅ then
     delete assignment in Q
  end
until #results > min_results or Q = ∅
output results
```

This algorithm describes the principle of all trading algorithms: Consider every profile available, select profiles that fit the transformed user query, and order them according to the degree of *conformity*, which is expressed by the function conf.

If no or not enough sources can be found, the trader simplifies the query by omitting attribute assignments. To decide which assignment should be deleted from the query, we use attribute ordering.

4.4 Conformity

The degree of conformity between a user query and a profile expresses to which proportion this profile fits the query. A value of 0 signals no conformity, a value of 1 represents full conformity. A provider is called *relevant* to a user query if its degree of conformity is grater than 0.

Consider the (dependent) attribute set {delivery_way, delivery_time} and the following user query Q:

delivery_way = online [1.0], email [0.5]
delivery_time = 0 [1.0], 1 [0.5];

The following table shows the profiles of four fictitious providers as far as they are concerned with the user query:

P_1	(fax,0)	(email,2)	
P_2	(online,0)	(email,0)	(postal,48)
P_3	(online,1)	(email,1)	
P_4	(online,0)		

It is obvious that P_1 is irrelevant, because this provider cannot deliver a document online or by email in less than or equal to one hour. It can quickly deliver by fax, however, the customer is not interested in this. Likewise, it is easy to see that P_2 is fully relevant.

Now consider P_3. It looks relevant, too. However, it is less interesting for the customer than P_2 because its delivery time is longer. According to the weights of the user query, a delivery time of 1 is exactly half as good as a delivery time of 0, so we set conf $(Q,P_3) = 0.5$.

Provider P_4 does not deliver by email, so it is less relevant, too. The customer has weighted both delivery ways 2:1, this means, P_4 lacks $\frac{1}{3}$ of the potential of P_2. Therefore, we set conf $(Q,P_4) = 0.\overline{6}$.

Let C denote the set of points in the attribute space, S the set of attributes in the user query, u_R^A the weighs for the value of attribute A in point R in the user query, and w_R^A the weights for the value of attribute A in point R in the profile. Considering the representations of the attribute values in the attribute space, we find the following expression for the degree of conformity (1):

$$\text{conf}\,(Q,P) = \frac{\sum_{R\in C}\prod_{A\in S} u_R^A w_R^A}{N} \tag{1}$$

To obtain the normalization factor N, we determine the normalization for each attribute A in the user query, using the set U_A of weightings for values of A (2):

$$N_A = \begin{cases} \sum_{z\in U_A} u & \text{if A is multivalent} \\ \max U_A & \text{else} \end{cases} \tag{2}$$

Then N is defined by $N = \prod_{A\in S} N_A$.

Applying this formula directly will lead to some problems, since finding all the points in the attribute space grows exponentially with the number of attributes n. But there is an easier way:

```
input     P,Q
N := 1; α := {ε}; ω :={1}          // ε denotes the empty sequence
for each attribute A in Q do
   α' := ∅; ω' := ∅; σ := ∅
   for each value a in A do
```

```
      let u be the weighing of a in Q
      let w be the weighting of a in P
      for each e in α do
          let v be the corresponding entry in ω
          if P contains the combination ea then
              append ea to α'; append v·u·w to ω' end
      end
      append w to σ
  end
  N := N · (sum σ if A is multivalent else max σ)
  α := α'; ω := ω'
  if α = ∅ then break end
end
s := sum of all entries in ω
output s/N
```

In this algorithm, α contains only those points in the attribute space in which the user query and the profile match (and ω contains the corresponding product of the weights). This is sufficient, because in any other point we have a resulting weight of 0. In case of independent algorithms, the calculation can be done in O(n).

5 Trader Federation

Both for technical and organizational reasons traders scale up only to a certain point. Further scaling usually relies on trader federation. For example, a trader may specialize in a organizational provider class (such as libraries) or in a technical class (such as providers dealing with documents about computer science).

We assume that each trader federation is hierarchically organized in the form of a tree, so that a trader can have several subordinate traders but at most one superior trader.

Every trader stores the addresses of its subordinate and superior traders; additionally, it holds the profiles of its subordinate traders. The profile of a trader is defined as the union of all profiles available to it; this profile is welldefined because of the hierarchical federation structure.

A trader forwards the user query to each trader which profiles is relevant for the user query. This *subsidiary query* is treated in the same way as a user query, with the difference that the query results are returned to the forwarding trader, which merges the results.

A query is only forwarded to the superior trader if the requested minimal number of providers cannot be found. This prevents the query from being immediately forwarded to the entire federation.

6 Related Work

Traders originally evolved from the naming services of telecommunication systems. A universal and formal definition of a trader originated in connection with the

idea of Open Distributed Processing (ODP), most notably in the ODP Trading Project [1,2]. The ODP Trader mediates between *Service Importers* and *Service Exporters*, so that *Service Offer* and *Service Request* match. The kind of the traded service is specified by the service type contained in both Offer and Request. The ODP approach was later extended to trader federations, by connecting *Interworking Traders* via a directed *Trading Graph*. The behavior of each trader is determined by the *Trader Policy*. The work resulted in the declaration of the ODP Trading Function satisfying the OMG Specification [4].

We decided not to use the ODP Trading Function for our approach. The huge and – for our intention – unnecessary complexity would slow down system development and complicate maintenance. The ODP approach gives no assistance for electronic commerce – which we need. Work in this direction can be found in [5] where the idea of the ODP Trading Function was merged together with the idea of an electronic market. Due to the universality of the ODP approach the proposed three-level architecture based on the Open Management Architecture (OMA) appears too complex for an information service on scientific literature.

Traders in the scientific arena are the Medoc Brokers [6] which enable the selection of sources that provide online documents or bibliographic data concerning computer science. Since the provider selection is based on metadata, the idea of the Medoc Brokers comes very close to our own approach. However, ranking of sources is only by order of estimated costs, and there is little cooperation among Medoc Brokers.

Learning from experience can be found in the *AI-Traders* [3]. The specification of types of services is based on conceptual graphs, which give an intuitive understanding of the intended service (*Intention*) independent from a physical representation (*Extention*). This is quite different from our metadata approach.

References

1. M. Bearman: *ODP-Trader*. In: Proceedings of the International Conference on Open Distributed Processing, Berlin, 1993, S. 37-51
2. A. Vogel, M. Bearman, A. Beitz: *Enabling Interworking of Traders*. In: Proceedings of the International Conference on Open Distributed Computing, Brisbane, 1995, S. 185-196
3. A. Puder, S. Markwitz, F. Gudermann, K. Geihs: *AI-based Tradering in Open Distributed Environments*. In: Proceedings of the International Conference on Open Distributed Computing, Brisbane, 1995, S. 157-169
4. OMG RFP5 Submission: *Trading Object Service*. 1996
5. L. Kutvonen: *Supporting Global Electronic Commerce with ODP Tools*. In: Proceedings of Trends in Electronic Commerce, 1998
6. M Dreger, N. Fuhr, K. Hroßjohann, S. Lohrum: *Provider Selection – Design and Implementation of the Medoc Broker*. In: Digital Libraries in Computer Science: The Medoc Approach, Lecture Notes in Computer Science, 1998, S. 67-78
7. M. Christoffel, S. Pulkowski, B. Schmitt, P. Lockemann: *Électronic Commerce: The Roadmap for University Libraries and Members to Survive in the Information Jungle*. In: ACM Sigmod Record 27:4, 1998, 68-73
8. Y. Arens, R. Hill, R. King: *Reference Architecture for the Intelligent Integration of Information*. http://mole.dc.isx.com/I3/html/briefs/I3brief.html#ref