Dynamic Support Service Selection for Business Transactions in Electronic Service Markets

M. Merz, T. Tu, W. Lamersdorf

Distributed Systems Group - Computer Science Dept. - Hamburg University

email: [merz|tu|lamersd]@informatik.uni-hamburg.de

Abstract

Business transactions in an electronic service market are expected to appear spontaneously, anonymously, and with varying requirements of infrastructural support. This support generally covers security requirements such as authentication, privacy, and public key certification as well as payment protocols or the demand for notary services. The latter two aspects are considered in particular in this paper. Due to the variety and ever-changing setting of such services, a flexible integration into electronic commerce infrastructures is required.

This paper proposes the embedding of *support services* into business transactions at run-time on the basis of a CORBA DII distribution platform. The architectural approach is described.

1 Introduction

Taking pattern from the definition given by Schmid [Schm93], an electronic market (EM) is a medium that realises market co-ordination at any time and place and increases transparency for all subjects involved in market transactions. It should reduce transaction costs in all three transaction phases: information gathering, contraction, and settlement. Electronic market systems (EMS) provide the computational infrastructure required by EMs. In this contribution, the EMS concept is restricted to systems that are not specifically designed to support electronically one single market segment such as stock exchange systems, but it rather assumes an arbitrary variety of on-line services across a multitude of business fields. Such a generalised EMS has not been realised yet for the Internet, since it lacks a full support of all transaction phases. Nevertheless, several research projects focus on distinct aspects and components of such an EMS like, e.g., electronic currency representations [JaWa95, ChFN88], notary and certification services [MML94b, LaNM94], or specific support of information gathering like traders, browsers or search engines [PMGG95, MMLT96]. An approach similar to the one discussed in this paper is the *InterPay* project, described in [CKPG+95] although Cousins et al. focus only on the integration of payment services.

1.1 Electronic Service Markets

An *electronic service market* is an infrastructure that allows free emergence and evolution of demand and supply of software services. The basic element of EM activities is a *market transaction*, which involves a customer and a supplier and comprises the delivery of a service against payment. Service suppliers who are immediately involved as a partner (a human user or a software entity) in market transactions are distinguished from third party service providers who support the market transaction. The latter are called *support services* that may act on a commercial basis similar to customer and supplier. However, their usage depends on the first level transaction. On a conventional market the purchase of goods may thus involve subordinated transactions to settle insurance coverage, to effect payment, to involve legal support for contractual matters etc. These activities depend on the actual purchase and may occur in varying combinations with different providers (such as lawyers, banks or insurance companies).

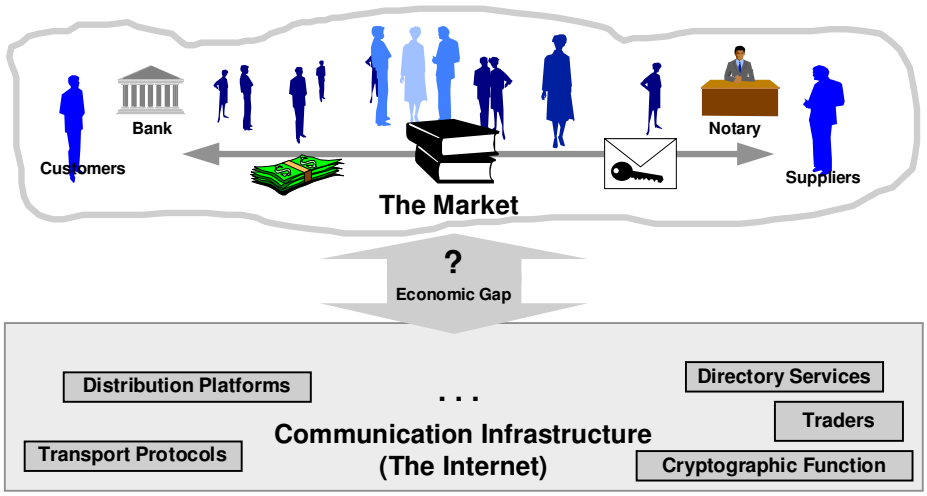

Fig. 1: The economic gap between distribution platforms and market mechanisms

If the EMS is to support market transactions in a similarly flexible way as it is possible in conventional markets, several shortcomings of existing distribution platforms have to be overcome:

- Often statically typed communication mechanisms prevent the ad-hoc utilisation of services in general. [MML94a] argues that platforms such as DCE only allow software components to interact that are tightly coupled at the level of interface and semantics. An EMS purely based on statically typed interfaces requires an a-priori standardisation of commercial services as well as support services. This approach restricts the free emergence of both commercial services and support services at an intolerable extent.

- On the other hand, environments such as the WWW or Java-based applications allow an immediate deployment of commercial services and consequently for clients to access them immediately. Client components are not semantically coherent to the server functionality here. They are rather used as *generic user access tools* such as Netscape© browsers or abstract machines that allow for remote execution of Java or mobile agent code. Although market requirements for the evolution of demand and supply are satisfied better, this approach lacks an architectural structure that assigns respective roles to the software components involved. This either leads to an individual configuration for each of the potential support services as known, e.g., from the Ecash payment service that is integrated into CGI scripts at the server side, or support functions that are tightly integrated into HTTP servers and clients as known from privacy and authentication enhancements in the case of Netscape tools.

Both approaches would principally allow to facilitate market transactions by closing the economic gap between the technical infrastructure and market processes (Fig. 1). However, both also imply high set-up costs at the same time if an alternative support service emerges as an attractive completion to existing ones. Both approaches thus delay innovation at the infrastructure level.

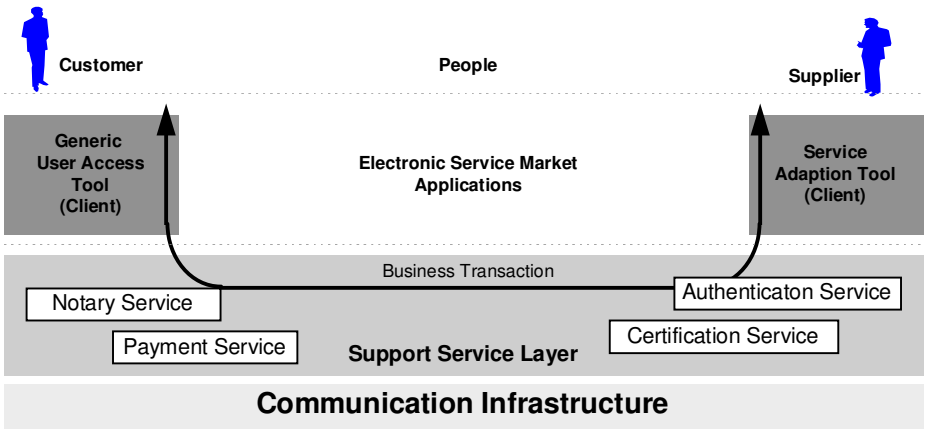

Fig. 2: The electronic market system

This paper focuses on the dynamic per-session integration of *support services*. An overall EMS architecture is assumed that generally enables customer and suppliers to carry out business transactions. Both customer and supplier are refered to as transaction *parties* in the following. The transaction itself is represented by a *session* (as communication channel) at the EMS level. It will not be established unless both parties agree on the underlying contractual conditions. The business transaction is performed by the invocation of *remote procedures* at the supplier's server. The task of support services is to enforce the agreed contractual conditions of both parties. A communication channel that ensures a secure communication is assumed as an integral part of the EMS. For the rest of the paper, notary and bank services are therefore selected as representative examples of support service.

The rest of the paper is organised as follows: Section 2 provides a realistic business scenario that is well applicable to derive requirements for an EMS architecture. Section 3 focuses on the role of support services in general and an architecture that allows for a run-time integration of support services into market transactions. Thereafter, the OSM architecture is presented which has primarily been dedicated to match the requirements identified before. The utilisation of CORBA DII and the Java-based integration of support services is sketched. Finally, implications of this architecture for a type-conforming client/server communication are discussed on the basis of a notary service example.

2 A Business Scenario

A customer intends to make a flight reservation. This transaction comprises a booking activity and the transfer of funds from the client to the service provider. Service providers such as travel agencies may sporadically appear on the service market with new service offers and they might not be trusted by the client. The provider is accessed sporadically as well by - from his point of view - untrusted clients. At the same time, one or both of the parties may require to carry out the booking transaction anonymously, i.e., neither of the parties themselves nor any third party is expected to unveil their identities.

In the case of a booking transaction, anonymity cannot be fully achieved since the service provided requires identifying data from the customer by its nature. However, one of the parties may either happen to repudiate an action that has been carried out as a part of the transaction, or - even more sophisticatedly - the parties may accuse one another of not keeping a contractually agreed specification of the transaction.

Due to the lack of trust, one party may suggest to involve a notary service in order to audit messages exchanged and to certify message delivery in a non-repudiative way. The other party has the option either to agree to this suggestion and to accept a suggested notary service or to refuse. The latter case will cancel the session establishment. The suggestion of a support service (such as a notary) may be made optionally by one or both parties.

Both parties maintain individual accounts at potentially different banks that, in turn, may support different payment protocols. In the scenario, the customer is assumed to maintain an Ecash account at ‘German Federal’ and a NetBill account at ‘Aachen Savings’, whilst the travel agency has a SET (Secure Electronic Transactions) at ‘Bill’s Bank’ and Ecash at ‘German Federal’. Obviously, there exists a common denominator (‘German Federal’) that allows both parties to carry out an Ecash-based payment. However, the proper selection of payment providers with respect to this information as well as the run-time integration of client software for payment services remains an open issue.

As a conclusion drawn from this scenario, two major requirements can be derived for a generic service market infrastructure if support services are to be flexibly integrated in a generic way:

- Classes, protocols, and instances of support services have to be given an agreed unique name in order to specify the requirements of both parties. A common *naming schema* is therefore required for the classification and identification of support services.
- A technique that allows to *dynamically access* appropriate support services in an arbitrary combination when a client/server session is established. This combination may vary from session to session. Since the implementor of the EMS infrastructure can not influence autonomous support service developers this access mechanism needs an abstract interface for all classes of services.

3 Support Service Requirements

3.1 The Classification of Support Services

Support services can be generally classified by the functions they supply. Some of these functions (payment and notary) have been involved in the business scenario. Others may be distinguished as complementary functions such as *quality assertion services*, which certify a distinct quality of service property to the client, or *protocol validation services*, which allow to restrict both client and server to a calling sequence (or *lifecycle*), originally specified by the server as a part of an augmented service description.

Many more support services may emerge that cannot be covered at design time by the EMS architecture. Therefore, a flexible naming schema is required that allows the registration of newly introduced *support service classes* at run-time.

Within support service classes, several *support service protocols* may be selected alternatively to perform the required function. The payment example of the scenario illustrated the options for client and server to involve payment service like Ecash, SET, or NetBill. These protocols are standardised, i.e., an agreed protocol identifier will lead to a well-known behaviour of the respective service. Also in the case of notary services several notary protocols may be available and have thus to be identified by agreed protocol names.

Finally, for a single support service protocol, a large set of individual *support service providers* may exist. 'German Fed' may be one provider of the Ecash payment service besides several more banks that support the same protocol.

This leads to a 3-level hierarchy of support services classes, protocols, and providers. The transaction parties may align their specification of support service requirements to this schema as illustrated below:

1. Customer:

```
  SS-Class:  Notary
^ SS-Class:Payment( SS-Prot: Ecash(    Bank: 'German Fed',
                                      Addr: 'mint.gf.de')
```

```
                  ∨ SS-Prot: NetBill(  Bank: 'Aachen Savings',
                                       Addr:'mint.as.de')
                );
```

2. *Supplier*:

```
  SS-Class:  Notary( SS-Prot: ISO-Notary(Name:'Voscherau',
                                         Addr:'henning.hh.de'))
∧ SS-Class:  Payment(SS-Prot: Ecash(     Bank: 'German Fed',
                                         Addr: 'mint.gf.de')
                  ∨ SS-Prot:  SET(       Name: 'Bill's Bank',
                                         Addr: 'mint.bs.de')
                );
```

These requirement specifications can be considered as predicates in conjunctive normal form since class-level requirements are combined by logical AND operators and protocols by logical ORs[1]. As access information, the provider level comprises a sequence of attributes for each provider specified. In the case of banks, names and internet addresses are shown. Further information is omitted for clarity matters.

After having identified a standardised classification schema for support services, a *matching* can be carried out by the unification of both terms. In principal, this schema is also applicable to the matching of more than two parties at the logical level. However, if a conflict resolution is required, one party has to be assigned a role as an initiator for this resolution process. To keep this simple in the following, only two-party matches are considered.

Some further conventions on the unification semantics are required:

1. How to treat a unilatereal requirement (one party requires a notary whilst the other doesn't even specify such a requirement)?
2. How to resolve different levels of specification? In the case of the scenario example, the client only requires *any* notary, whilst the server specifies it at the provider level (e.g., "Voscherau")?
3. How to resolve conflicting requirements (e.g., if the parties specify different notaries)?
4. How to resolve the contrary case: *both* parties only specify that *any* notary is to be involved, yet none of them names a distinct provider?

Since the number and types of support services and parties involved in a business transaction can vary greatly from one application to another, a generic mechanism is needed to resolve these possible conflicts when matching the participants' requirements. In general, the resulting requirements returned by the matching process have

---

[1] The semi-formal notation of this example is chosen to illustrate some possible requirements of customers and suppliers. As input for the matching engine, a conversion to a fully formal notation needs to be performed.

to be such that all participants' requirements are fulfilled. Such a mechanism can work as follows:

For the *first case*, a semantics appears to be suitable that forces the other party (with less specific requirements) to agree if it can employ the support service required by the more specific party. That means, an undesired service would have to be refused explicitly by a negative statement. This "agree-if-not-specified" semantics seems to be practical, but other semantics could be imposed on the matching process (as meta-policies) as well.

The *second case* can be understood as an implicit agreement of the less specific party to the requirements of the other. This policy would oblige the client to accept the notary selected by the server.

The *third case* will lead to a rejection of the session if no additional means to resolve the conflict exists. Therefore, if the transaction should still take place, some kind of meta-policy, also called arbitration policy, would have to be used. An arbitration policy may state, for example, that with respect to notary services, the client's requirement can be overwritten by the server's.

The *final case* may be resolved by picking the *first*, a *random* or a *default* notary service that can be employed by both parties. However, these policy management assumptions may either be a static part of the infrastructure or themselves subject to run-time configurations by means of meta-policies. The latter case would require a dedicated policy management function which is not the issue of this paper. Therefore, these policy management approaches will be assumed for the following sections.

Due to these policy assumptions, the resulting requirements for both client and server will look as follows:

```
  SS-Class: Notary( SS-Prot:ISO-Notary( Name:"Voscherau",
                                        Addr:"henning.hh.de") )
^ SS-Class:Payment( SS-Protocol: Ecash( Bank: 'German Fed',
                                        Addr: 'mint.gf.de') );
```

After such a matching has been carried out successfully, the actual session between client and server can be established.

3.2 Session establishment and interaction

After having introduced a naming schema for support services, a suitable representation for requirement specifications, and a semantics for matching the requirements of both parties, the actual access to support services for client and server applications needs to be investigated.

Binding to support services

An RPC service is assumed as the underlying communication mechanism. In the scenario, the supplier is assumed to offer the conditions under which a session establishment between the customer’s client and the supplier’s server is accepted.

Therefore, a transfer mechanism is required that communicates this information from server to client. This *service representation* does not only contain a specification of the service interface but also arbitrary additional information on the server's semantics. Among these data objects, also support service requirements from the server's side can be embedded. In COSM, the service representation is used as a carrier for these description objects [MML94a].

As a dynamic user access tool, the *generic client* has been configured at start-up time with the customer's support service requirements. Therefore, it only needs to acquire the service representation in order to perform the matching process discussed above. After the resulting requirements have been obtained, the actual connection to support service providers has to be established from both the client and the server side. As far as the EMS infrastructure is concerned, a dynamic binding mechanism is required for the run-time access of support services.

For this purpose, the usage of *proxy objects* as mobile client components appears suitable to satisfy the binding requirements (Fig. 3). On the client application's side, communication with proxies takes place across a standardised interface that allows proxies for any support service class to interact with the client application. Regarding the support service, the specific (notary or payment) protocol is abstracted away. The same applies to the server application's side: here are proxy objects acquired likewise in order to communicate with support services. They are dynamically bound corresponding to their client side counterparts. Again, interaction between the server application and server-side proxies is standardised as well.

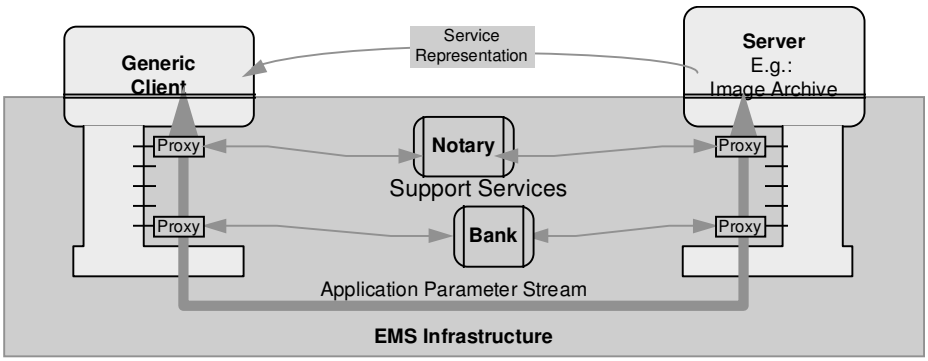

Fig. 3: A configured session with dynamically bound support services

Interactions between client, server and support services

Different support service protocols require an individual integration into the client/server data exchange: beyond the actual RPC parameter transfer, communication links exist between client and server proxies and with support services. This communication may occur *before* RPC parameters or results are actually transferred or *thereafter*: In case of a notary service, hash values of parameters are generated by both parties before and after parameters and results are transmitted. On the other

hand, in the case of a payment service such as Ecash, electronic coins can be inserted into the parameter list such that no immediate interaction is required between the client and the bank.

If the application parameter stream could be utilised for the data transfer between corresponding proxies, a flexible link could be established between them. By embedding proxy parameters into application parameters lists, they could be transferred as 'piggyback' data. This applies for example to the exchange of session keys between notary proxies on the client and server side.

However, this approach requires, first, an application parameter representation that is inspectable at run-time by support service proxies and, second, the posibility to read or append elements of the parameter lists. Such operations are not feasible if application parameters are statically typed. Since distribution platforms such as DCE or CORBA define a transfer syntax that neither assigns type-tags nor names to parameter values, a dynamically typed transfer syntax is required. Here, only CORBA values of type *Any* allow to be interpreted and therefore *intercepted* when transferred at run-time.

Consequently, a suitable solution is to utilise the CORBA DII at both client and server sides. The applications provide RPC parameters and results as *named value lists* (actually *named value trees* in the EMS framework). If this representation allows each proxy to insert individual data values in a standardised way, they are enabled to communicate with their corresponding counterparts at the other parties side.

To support an agreed parameter embedding technique, the various options of support service classes and protocols need to be reflected by the schema of named value trees: additional 'slots' are used by proxy objects next to application parameters. The resulting value tree for a client which is supported by a notary and a payment service is shown schematically in Fig. 4. Here, the distinguished parameter `SuppSvc` refers to a list of selected support service classes, each represented by another subordinated list. This list contains protocol-specific data values that are exchanged between proxies of the respective class.

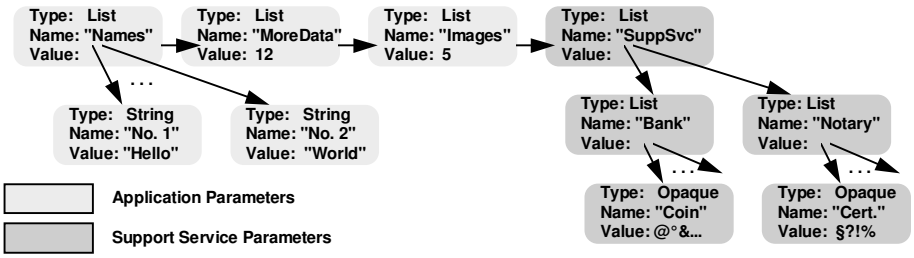

Fig. 4: Parameters transfered between client and server

Type compatibility for dynamically augmented value lists

As parameter and result values are extended by proxies, they become a *subtype* of the previous version. This can be demonstrated on the basis of *value lists*: the original value list $\{P_{S1},... P_{Sn}\}$ (as expected by the server) is extended by additional parameters $\{P_{N1}\text{-}P_{Nn}, P_{B1}\text{-}P_{Bn}\}$ from the client's notary and bank proxy (P_{Nc} and P_{Bc}, respectively). When transfered to the server, this extended list represents the interface type actually requested by the client. Its parameter type is a subtype of the interface provided by the server: $\{ P_{S1}, ... P_{Sn}\}$. Due to the contravariance for parameter types, this implies that the provider's signature actually is a subtype of the requested one, i.e., both are compatible (see [CaWe85]):

Provided Signature (Sig_S) **<:** *Requested Signature (Sig_C)*

$$\texttt{foo } \{ P_{S1}, \ldots P_{Sn}\}\texttt{: } \{ R_{S1}, \ldots R_{Sn} \} \texttt{ <:}$$
$$\texttt{foo } \{ P_{C1}, \ldots P_{Cn+j} \}\texttt{: } \{ R_{C1}, \ldots R_{Cn-i} \} \qquad \texttt{with i,j >= 0}$$

By extending the parameter list follows:

Sig_S <: Sig_C <: Requested Signature with extended parameter List (Sig_{C+})

$$Sig_c \quad \texttt{<:} \quad \texttt{foo } \{ P_{C1}, \ldots P_{Cn+j}, P_{N1}\text{-}P_{Nk}\ P_{B1}\text{-}P_{Bl} \} \texttt{ : } \{ R_{C1}, \ldots R_{Cn-i} \}$$
$$\texttt{with i,j, k, l >= 0}$$

Correspondingly, proxies from the server's side may extend the result list by additional data values. Such an extension implies a subtype of the originally provided signature:

Sig_{S+} <: Sig_S <: Sig_C

$$\texttt{foo } \{ P_{S1}, \ldots P_{Sn} \}\texttt{: } \{ R_{S1}, \ldots R_{Sn}, R_{N1}\text{-}R_{Nk}\ R_{B1}\text{-}R_{Bl} \} \texttt{ <: } Sig_s$$

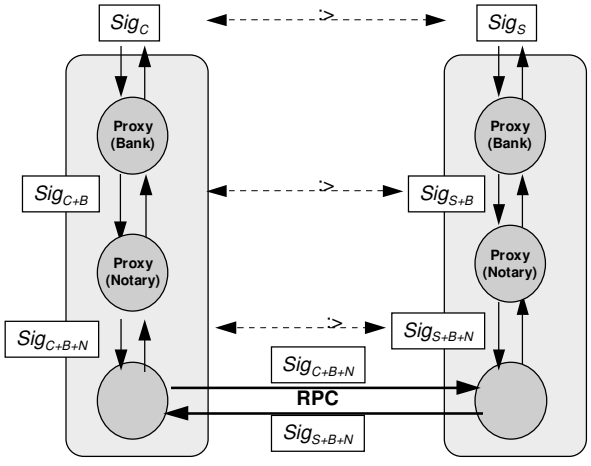

Fig. : Subtype matching with extended parameter and result lists

Finally, a combination of both extensions will lead to conforming interfaces types of client and server due to the transitive character of subtype relations:

Sig_{S+} <: Sig_{C+}

The original subtype relationship between client and server is still given (see Fig 5). Since parameter values are identified by their names rather than by their position within the value list, the original list order may also be rearranged arbitrarily by intermediary proxies without affecting the identification of distinct values.

4 OSM - An Electronic Service Market Infrastructure

The *COSM* (Common Open Service Market) project at Hamburg University aimed at establishing a suitable EMS architecture[OSM96]. However, current investigations in the course of the follow-up project *OSM* (Open Service Model) lead to an integration of market infrastructure components such as support services, traders, and matching mechanisms that deal with protocol specifications.

Beyond other functions, the OSM project aims at implementing the dynamic support service integration that was discussed above. For OSM, *Java* has been selected as the implementation language and CORBA as the architecture framework. The usage of Java facilitates dynamic loading and binding of proxy objects whilst CORBA DII provides a flexible parameter transfer mechanism that supports the value exchange between corresponding proxy objects.

In OSM, a *service profile* is used as an agreed data structure in order to specify *service interfaces and semantics*, to capture session specific *context information*, to provide a *contractual basis* for the business transaction, and to *store persistently* this information at the generic client side.

A service profile can also include *service type specifications* for trader import and export functions [ISO-ODP95a] and for catalogue services. It further contains descriptions of *negotiation protocols* as well as *service access protocols*. In the first case, service selection is supported by *brokers*, which arrange client-server bindings on the basis of standardised negotiation protocol. The latter describe *valid interaction protocols* between clients and servers after a session has been established.

5 Summary and Outlook

This paper presents a flexible technique for the integration of (trusted) third parties into an electronic commerce session in order to support secure business transactions. To achieve this, a naming schema and a flexible binding mechanism are required for support services. Concerning both aspects, a solution was suggested that allows for the independent development of client and server applications, EMS infrastructure components, and arbitrary classes of support services. The presented approach doesn't hinder a free evolution of all software components involved (clients, servers, and support services).

However, several open issues still need to be addressed:

- *Security*: in an anonymous and competitive environment, malicious proxies or support services may occur. The scenario sketched above lacks a security mechanism that ensures a binding only to validated, certified, and therefore trusted proxies. While anonymity, autonomy, and competition dominate the application level, security requirements should thus govern the infrastructure layer.

- Matching and conflict resolution mechanisms can turn out to be much more sophisticated than a unification mechanism based on first-order predicate logic: what if both transaction parties don't have a payment protocol in common, but there exists a third party agent that is able to provide a clearing service between, e.g., Ecash and SET? This cannot be derived from the requirement specification because of several reasons: first, the EMS as a distributed system is far too decentralised for providing matching information of this kind in a consistent way. Second, the number of possible matches facilitiated by clearing services cannot be captured by a limited requirement specification as the number of payment protocols and their interrelations grow.

A possible solution to this general problem could be to factor out the requirement matching task to a general *policy management service*. Different approaches to deal with this and other questions like those mentioned in Section 3.1 are currently being examined as a subject of research at Hamburg University.

6 References

[CaWe85] L.Cardelli, P.Wegner: "On Understanding Types, Data Abstraction, and Polymorphism". In: *ACM Computing Surveys*, 17(4), 1985, S. 471-522

[ChFN88] D. Chaum, A. Fiat, M. Naor: "Untraceable Electronic Cash". In: S. Goldwasser, Ed., *Proc. CRYPTO '88*, Springer, Berlin Heidelberg New York, 1988

[CKPG+95] S.B. Cousins, S.P. Ketchpel, A. Paepcke, H. Garcia-Molina, S.W. Hassan, M. Röscheisen: "InterPay: Managing Multiple Payment Mechanisms in Digital Libraries". In: Proc. 2nd Annual Conference on Digital Libraries, Austin, Texas, USA, June 11-13, 1995.

[ISO-ODP95a] ISO / IEC JTC 1 / SC21: "ODP Trading Function". Draft International Standard 13235, 1995

[JaWa95] P. Janson, M. Waidner: "Electronic Payment over Open Networks". INFORMATIK - Zeitschrift der Schweizer Informatikorganisation, 1(3) Juni 1995

[LaMN94] C. Lai, G. Medvinsky, B. C. Neuman: "Endorsements, Licensing, and Insurance for Distributed System Services". In: *Proc.*

2nd ACM Conference on Computer and Communication Security, 1994

[MML94a] M. Merz, K. Müller-Jones, W. Lamersdorf: "Service Trading and Mediation in Distributed Computing Systems". In: L. Svobodova, Ed., *Proc. 14th ' nternational Conference on Distributed Computing Systems'*, Poznan, Poland, IEEE Computer Society Press, 1994, S. 450-457

[MML94b] M. Merz, K. Müller-Jones, W. Lamersdorf: "Trusted Third-Party Services in COSM". In: *EM - Electronic Markets*, Institute for Information Management, University St. Gallen, Switzerland, Heft 12, Sept. 1994

[MMLT96] S. Müller, K. Müller-Jones, W. Lamersdorf, T. Tu: "Global Trader Cooperation in Open Service Markets". In: *Proc. Intl. Workshop 'Trends in Distributed Systems'*, Aachen, Oct. 1st-2nd 1996, Springer, Berlin Heidelberg New York, 1996

[OSM96] Home Page of the Open Service Model project at Hamburg University: http://osm-www.informatik.uni-hamburg.de, 1996

[PMGG95] A. Puder, S. Markwitz, F. Gudermann und K. Geihs, "AI-based Trading in Open Distributed Environments". In: *Proc. IFIP International Conference on Open Distributed Processing*, Brisbane/Australia 1995

[Schm93] B. Schmid: "Electronic Markets". in: R. Alt, St. Zbornik (Eds.) Electronic Markets Newsletter, 1993, S. 3-4