International Journal of Cooperative Information Systems
© World Scientific Publishing Company

SUPPORTING ELECTRONIC COMMERCE TRANSACTIONS WITH CONTRACTING SERVICES

MICHAEL MERZ, FRANK GRIFFEL, TUAN TU, STEFAN MÜLLER-WILKEN
HARALD WEINREICH, MARKO BOGER, WINFRIED LAMERSDORF

Distributed Systems Group
Department of Computer Science, Hamburg University
Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany
http://vsys-www.informatik.uni-hamburg.de

Based on the specific characteristics and requirements for an adequate electronic commerce system support, this article gives an overview of the respective distributed systems technologies which are available for open and heterogeneous electronic commerce applications. Abstracting from basic communication mechanisms such as (transactionally secure) remote procedure calls and remote database access mechanisms, this includes service trading and brokerage functions as well as security aspects including such as notary and non-repudiation functions. Further important elements of a system infrastructure for electronic commerce applications are: common middleware infrastructures, componentware techniques, distributed and mobile agent technologies etc. As electronic transcations enter the phase of performance, increasingly new and important functions are required. Among these are: negotiation protocols to support both the settlement and the fulfillment of electronic contracts as well as ad-hoc workflow management support for compound and distributed services in electronic commerce applications. In addition to an overview of the state of the art of the respective technology, the article briefly presents some related projects conducted by the authors jointly with international partners in order to realize some of the important new functions of a system infrastructure for open distributed electronic commerce applications.

Keywords: Distributed Systems, Electronic Commerce, Electronic Contracting, Componentware, Workflow Management, Service Trading, Brokerage, Negotiation

1. Introduction

As an area of research, Electronic Commerce has various facets that span from the economic and legal infrastructure over software standards and platforms to horizontal applications (which are specific for a certain function) and vertical applications that address the needs of a certain business sector. In this article, we will present electronic contracting as a horizontal transaction support function. Electronic contracting can be defined as the complete process that is required to achieve a legally supported business relationship that is accompanied by an electronic contract as the common and neutral representation for all obligations the involved parties agreed on.

1.1. Centralized vs. decentralized architectures

Today, many users of large computer networks demand a re-centralization of enterprise computing systems in order to reduce roll-out efforts and maintenance costs. This development can be considered as a sober response to the idea of distributing any service at any time on any kind of heterogeneous hardware and operating system environment. Today, certain "low-level" services are accepted as inherently distributed such as DNS, NFS, HTTP, SMTP, etc. On the other hand, there are many others that were expected to be distributed applications in the early 90ies and before: distributed databases, shared editing, application-level extensions to the telecommunication infrastructure. However, they did not succeed in day-to-day practice by now. Why that?

One may argue that the first of these services are historically better understood, but there is another reason for the lacking success of the latter: the complexity of their respective specifications. A distributed database is not only itself a complex application but also the data models on top require immense conceptual and technological integration effort. If there exists a centralized alternative that could be followed without facing prohibitive costs — this option would usually be chosen.

For a couple of years, however, the Internet established a drastically decentralized communication platform that flattens hierarchies, overcomes organizational borders and liberates small companies and individuals from high investment costs for communication access. This development stimulates the cooperation between business partners and their heterogeneous, proprietary software systems.

Finally, it is trivial to state that organizations are distributed across cities, regions, and countries. Since they coordinate the exchange of goods, services, and payments across their organizational boundaries and therefore across long distances, we find a natural need for a correlated support of electronic commerce applications by distributed systems. In this area, the distribution of applications is not only an option but a precondition for commercial life.

However, interorganizational business relationships coincide with contractual relationships, as has been elaborated.[1] To overcome delays and bureaucratic obstacles, we propose *electronic contracting* as a supporting service that helps enterprises to

- identify business partners,
- match their individual offer specifications with complementary ones from other market participants,
- negotiate conditions and contractual terms,
- collectively sign contracts, and finally,
- execute obligations and actions that are defined in the contract.

For this reason, we will first analyze the structure of an electronic commerce transaction in more detail and afterwards present the supporting services mentioned above.

1.1.1. Modeling commercial transactions

To provide a systematic classification of electronic commerce technology, two viewpoints could be taken: First, a distinction between businesses, consumers, and public authorities could be chosen for the respective roles of buyers and sellers of goods and services.[2] However, if we consider single persons as legal entities thus representing a market participant equal to a company, the borders between these categories will blur. We therefore consider any market participant as a "business". Instead, we will follow the *phase model* for commercial transactions as the guidelining pattern to decompose architectural elements of an electronic contracting service.[3] The phase model consists of three main phases:

- In the first *information phase*, market participants offer product specifications, search for possible transaction partners, compare product specifications and prices, and evaluate offers.
- Then, after an initial contact has been established between some market participants, respective (service) offers and counter-offers are exchanged during the so called contract *negotiation phase*. This negotiation process may either lead to a situation where agreed terms and conditions have been reached or the negotiation is abandoned.
- In case of a contract establishment, first all participants commit their participation in the contract with their respective signatures, then the agreed assets are exchanged during the contract *performance* (or: *execution*) *phase*. The time-span of this phase may reach from a few seconds up to several years.

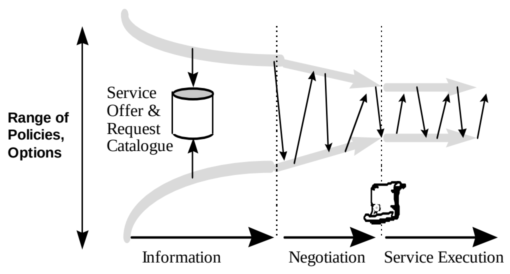

Fig. 1. Business Transaction Phases.

Following these phases, a clear separation of services can be given that are required for an electronic marketplace:

(1) *Information phase*: This phase may be supported by (computer) functions like online catalogues, search engines, or banner advertising.
(2) *Negotiation phase*: Here support for telecollaboration, negotiation protocols and strategies may be used to achieve an agreement.

(3) *Execution phase*: During this phase, workflow management, business process integration among market participants, electronic payment systems, EDI-based message exchange functions, etc. may be provided in order to support the automatic execution of electronic commerce applications.

In between these phases, the following additional functions may be required:

- *Brokerage support* in order to select and match respective offers and inquiries, to form a (service providing) consortium or to set-up the negotiation session for all parties of the commercial transaction
- *Signing support* to enter the execution phase by establishing a contract and ensuring for all parties to sign it. This process may be supported by trusted third parties such as certification authorities or electronic notaries.

In order to keep the model simple, yet without unrealistic abstraction, we consider any good as a service:

- A *payment* is a service provided by the customer. The result of the service is the transfer of data which is interpretable as a transfer of a value. This might either be an electronic coin or the authorization for a funds transfer between two bank accounts.
- A *tangible good* can be represented in the system as a service: It is selected, ordered and paid electronically and even the physical delivery is accompanied by a range of services and data communications that may be used by the commercial parties (transfers of electronic EDI documents, access to information on the delivery state, etc.).

1.2. Organization of the article

According to the progress of a commercial transaction introduced above, the next section starts with a brief introduction of a *catalogue* service that provides the common basis for all contracting activities. As the next step, the *brokerage* process is described as a function that establishes contract proposals on behalf of the contracting parties. In order to *negotiate* contract terms and conditions, the required collaboration support is described afterwards. Contract *signing* and *execution* are subsequently addressed in sections 5 and 6. The article concludes by reflecting on a component-oriented view of the described software systems.

2. Catalogue Services

To inform customers about the range of products and their specifications, catalogue services are used as a shared information base for both vendors and customers. A catalogue service also establishes a comfortable front-end for activities that are carried out under the execution phase (such as payment and product delivery). Internally, catalogues are updated by the supplier's stock management system to keep the information displayed in sync with the physical warehouse.

Today, a catalogue service is deployed by an individual supplier who intends to provide access for the customer to the range of tangible or intangible ("soft") goods. For small vendors, a catalogue may be hosted by a third party in the same way as web servers are hosted by an Internet Service Provider. In this case, an individual vendor remains responsible for his own "shop" in terms of presentation and product data. A unified settlement of payments and possibly the delivery of soft goods, however, is centralized by the shopping mall provider.

Consequently, providers of catalogue services tend to further break down the hurdle for offering goods in a mall system by allowing vendors to enter single offers into the catalogue: this leads to an offer database, that allows competing market participants to register their offers in a suitable category of the offer database.

This concept has already been addressed several years ago by the ODP trader service,[4] that not only suits for storing (*exporting* in ODP lingo) product specifications but also service specifications in a formal sense: services are understood as instances of a *service type* that includes *service attributes* and the *interface type.* A trader additionally provides the matching of offers and inquiries. Later-on, this technology has been incorporated into CORBA standardization as the trader Object Service.[5] Before focusing on the trader service in the following section, we will first illustrate the catalogue information model: three kinds of data are stored:

- *Participant profiles.* These contain information on the ID, name, and address of the participant. Also online addresses and individual communication preferences (such as encryption, authentication, non-repudiation, etc.) are stored here. Profiles can be browsed and queried by other market participants.
- *Offers.* Participants store offers into the catalogue database. Offers mainly consist of dynamic feature sets that represent product specifications as name-value pairs.
- *Contract templates.* Contracts bind together parties (recruited from market participants), obligations, and other legal clauses. In the state of a template, a contract in incomplete, i.e., parties are not completely defined and offers only specified but not yet matched to an offer entered into the catalogue.

Catalogues can be browsed and queried through all three entry types. By using a dedicated tool — the *contract editor* — users may step-by-step compose complete contracts from scratch or by re-using contract templates as well as existing offers and profiles.

3. Service Brokerage

The next step in any business transaction process is the location and selection of an appropriate business partner. Based upon a more or less precise concept of the attributes the customer is interested in, a catalogue service will be contacted to search for suitable offers. Apart from a specific description of the offered good itself, such offers most commonly contain information on business conditions and how to contact the supplier, negotiate where appropriate and finally complete the business transaction.

Various catalogue services — or more general — sources of information are used in every day life. Possible alternatives range from plain *white pages* services over *yellow pages* services and professional publications to mail-order catalogues. Additionally, more and more third party services offer their customers the convenience of only having to formulate certain criteria and a price range of the good they are interested in.[6] It is then the duty of the service provider to locate a supplier that has the most suitable offer with respect to the formulated query. It is primarily due to the relatively high transaction costs (call the provider, formulate a query, be charged for the service etc.) that this optimised way of service location is yet only available for more valuable goods. And it will be especially this area, where commerce will definitely benefit from getting "electronic"!

The process sketched above best maps into the commercial transaction's information phase and here especially into the brokerage support. Customers first gather information by means of catalogue services, find goods or services they are interested in and then use a brokerage facility to locate the optimum supplier to deliver the good or offer the service. Using the information they receive from the brokerage facility, the can then start a conversation with the supllier to finalize the business transaction.

All aspects of an "electronic service market" are to take place on–line.[7] This is especially demanding with respect to the information phase and here it consequently means that one will need to have an efficient, reliable and economical means to locate and access services in highly distributed systems. To accomodate these goals, it is profitable to integrate methodology as developed in the field of ODP (open distributed processing) and CORBA (common object request broker architecture), where the type–safe location and mediation of services has become an important aspect.

3.1. Service mediation in distributed systems

The introduction of low- to medium-cost computer systems in the mid-eighties led to an increasing interest in distributed computing systems. Where centralization had been the most important "weapon" to gain cost effectiveness in the computing business, new buzzwords were *scalability* or *heterogeneity.* Terminals were substituted with desktop PCs and powerful workstations entered the midrange computing market. It was then, when the need arose to find reliable tools to create software, that could be spread over a whole group of machines, sometimes thousands of miles away from one another.

Various approaches were made to solve the problems, of which the introduction of so called *middleware* plattforms may have caused the deepest impact.[8] Middleware refers to a software layer that mediates between the application and the actual operating system and consequently abstracts from peculiarities of the operating system, the computing system or the network it runs on. Additionally, middleware plattforms offer supporting services (authentication, distributed filesystem, time synchronization, etc.) usually needed by a broad range of application systems. Of the attributes that constitute a middleware plattform, *openness* is perhaps the most important. Openness in this context means the availability of protocol specifications to anyone, thus enabling the integration of what system soever.

The advantage of the middleware approach for the application developer is that he can now more easily port software to different plattforms and integrate them without putting much effort into dealing with varying plattform issues. The advantage for the computer user was that she could now pick the system best suiting a certain environment or demand while still being able to be part of a large system — which becomes most important with workflow-like execution of a contract (see sections 6 and 7). New workstations could be integrated or removed as necessary, a feature that led to very cost efficient computing system solutions.

An important concept in a distributed system is that of *service types*. All services that are available in a distributed system are instances of an explicitly or implicitly defined service type, thus making it possible to classify all existing services and support course-grained queries on such classes. A service type generally consists of a name, a set of properties and a list of interfaces a service conforming to this type has to offer. A service property could be the maximum number of pages a print service is able to process per minute or the printing resolution it offers. The service type does not contain actual values. It merely defines the type of value a property may have, e.g. "string" or "unsigned integer". The interface types define the operational interfaces a service offers. Each interface type lists a number of *operation types* where each operation type can contain zero or more *data types*.

Based on formal rules, service types can be organized in a type hierarchy where specializations of a type are called *subtypes*, while generalisations are named its *supertypes*.[9] If a service instance belongs to a subtype, it may — at least formally — be used, where a service of its supertype is required. This means, that an instance of a subtype will offer the same operations that can be called with the same number and type of arguments. Obviously, an instance of a subtype will not automatically have the same property values, so unless the service is stateless (e.g. mere calculation based on operation arguments), this fact will have to be taken into account.

It is obvious that larger scale distributed systems raise the problem of finding appropriate strategies of locating service instances. While more static systems will allow for special service location tables, systems with more dynamicity will demand more sophisticated means. As with the real life example given above, the easiest approach will be that of "white pages", in distributed systems terminology called *catalogue services*. A catalogue service holds entries with name and type of a service along with a reference to its location. If there are more service entities of the same service type, it is the callers duty to pick one, using name and location as decision aids. Simpler catalogue services will offer a flat list of entries, more sophisticated ones might have a hierachical structure. While using catalogue services might suffice for smaller, or less dynamic environments, it is not practical for extremely dynamic environments like that of global scale "network-markets". Here, numerous instances of a service type might arise and their name and location would no longer be adequate to find the best offer. It is the notion of *service mediation*[10] that fills that gap.

The main task of a service mediation facility is to manage and mediate services in an open distributed systems. To accomplish this, the mediation facility provides functions to classify and store service offers and to locate certain service offers that are stored within an offer space. Generally, the service mediation facility is referred to as a *trader*, a service provider is called *exporter*, whereas the service client is called *importer*.

The relation between trader, exporter and importer is most commonly refered to as the “trading triangle”. The exporter first contacts the trader to advertise its service. To do so, it sends a service type reference, that is, the category the service belongs to, values of all service properties along with a reference to its call interface in an *exporting request* to the trader. The trader stores the service advertisement in its *offer space*. When a client is looking for a service of a certain type and certain characteristics, it formulates a query that contains the sought service type along with the selection criteria and sends it in an *import request* to the trader. The trader scans its offer space for offers of the required service type and returns all offers of that type that additionally match these criteria. Finally, the client uses the interface reference stored in a returned service offer to initiate a conversation with a service provider.

The TRADEr, developed as part of the “TRADE” (TRading And coorDination Environment) research project at the University of Hamburg, was one of the first trading facilities available.[11] It was designed as an enhancement to the DCE middleware environment and thus could be used by any DCE-aware application. The TRADEr contained all features mentioned above, including a type-management component, a service selection and management component and an access control module. Additionally, the TRADEr included a trader federation facility,[12] thus providing the means for partitioning the service offer space and making it possible to deal with scalability problems in larger environments.

Trading services have been formally defined for both ISO ODP[4] and OMG CORBA[5] and academic as well as commercial implementations are currently available. Still, today´s trading services have conceptual deficiencies that have to be handled, before suiting the needs of an electronic service market.

3.2. The brokerage facility — n-party trading for electronic service markets

So far, all service types and service instances were implicitly assumed to be “atomic”. However, this represents no realistic picture. It is more likely that services in any serious application scenario will be aggregations of, or at least contain, multiple service instances themselves. A real world “print service” for example, while offering one service to its customers, essentially would consist of a printing service, a binding service or maybe even a layout service. To take this into account, various enhancements have to be made to the notion of service types as well as to service mediation methodology.

First of all, a new feature within the brokerage facility is the introduction of *service type templates*. In contrast to traditional service types, templates allow the nesting of service type definitions by means of inclusion of service type references in addition to operation- and property types. This allows a more realisic modeling of services and is the basis for an efficient service mediation facility in electronic market scenarios.

Conceptionally, a service type template offers an operational interface like any service type would, while at the same time internally consisting of or containing references to multiple service types or type templates. Basically, this can be realized in two ways: a service type template can be an aggregation of the included service types, only combining their operational interfaces and properties; or it can be a new, enhanced type with an own interface and own properties. While the latter better resembles reality (where the whole is more than just the sum of its parts) it still is to be investigated how to adapt existing type theory (espec. derivation rules).

A second important novelty is the definition of service type *factories*. A service type factory is a parametrizable service type where an actual service can only come into existence, when certain thresholds are passed. This can, for example, be the number of clients that have to state their interest in a service type before a service can be instantiated or the number of resources of a certain kind that are necessary to realize some service.

As an introducing example, one might think of car production. When manufacturing cars, there will always exist certain thresholds from where on initial investments will be regained and the business starts to be profitable. It is thus the aim of any car manufacturer to sell as many units as possible of a certain model. On the other hand it is natural only, that customers always tend to show some individualism by demanding extras in a car and they will prefer the manufacturer that is able to deliver them.

In an online service market context, one could formulate a service type factory that defines the number of customers necessary for a profitable production. Another example might be that of an auction that can only take place with a minimum number of bidding parties but will no longer work when the number of bidding parties exceeds some threshold.

In both cases enhancements will have to be made to the service mediation facility itself. Where before the service mediation ended with the delivery of service offer references to the client, now the process is enlonged: in the first case, an offer can only be delivered when a certain number of clients have stated their interest. The clients will have to be committed to the query for a defined time and the query will have to be postponed until enough clients have been “booked”. In the second case, services may be added or substituted even *after* the filled template has been delivered. Both are capabilities not available with existing trading facilities.

3.3. Brokerage in the field of online service negotiation

The brokerage facility will be of particular interest for service negotiation and configuration. As described in more detail in the next section, brokerage facilities can be used to fill so called protocol templates (another form of service type templates) and thereby support agents in finding appropriate negotiation parties. Here it is essential to the agent, that the process of filling a template is invisible to him and as mentioned above, it will be the duty of the broker to locate and commit service providers to take part in a negotiation.

4. Service Negotiation

Negotiation is the process of reaching an agreement for a service specification. This may take place either out-of-band, by letting market participants negotiate without electronic means, or it may be done on-line. In this case there are several stages of automation possible:

- *Using collaboration tools.* In this case, human users are involved in the negotiation process. They use, e.g., a shared-editing tool that allows them to concurrently edit a document in a consistent way. The negotiation is free-form, i.e., there are no restrictions for the order of document accesses or the structure of the document.
- *Using negotiation protocols.* In this case, either human users or software components participate in the negotiation. The negotiation subject is still unstructured, i.e., the participants "know" how to deal with it. The ordering of document accesses however is formalized and parameterized, .i.e., a negotiation protocol is applied to specify which party delivers which information at which stage to whom. The negotiation can be understood as a workflow process that is driven by a predefined process description.
- Using *formalized conversations* to further structure the negotiation protocol. Speech act theory (Specifically, the Knowledge Query and Manipulation Language, KQML)[13] provides a linguistic means to define formalized messages that relate - in the case of negotiations - to concepts such as "offer", "reject", "propose", "accept", etc. This further helps to tailor the involved software systems for the specific application of negotiation support: it may, e.g., react in a different way when it receives an offer instead of a proposal.
- Finally, the *complete negotiation process* may be automated (and therefore delegated to "autonomous" software components) if the ontology for the negotiation subject has been standardized as well. In this case "speed" and "price" are features that a software component is able to reason about. Therefore, AI technologies are applied in this area for knowledge representation and for applying policies that have been defined to control negotiation strategies. Such an intelligent software agent is now capable at least to estimate "price" and "speed" and to trade-off their values in a reasonable way.[14]

In today´s real world, automated negotiation is not used so far for the following reason: Only if the service specification is kept simple (i.e., only a few Quality of Service attributes), a strategy module can be practically used for negotiating them. The more complex the specification becomes, the more effort needs to be spent for implementing policies and strategies for the agent. If a simple specification is sufficient, the service can be considered as a commodity on the other hand, i.e., a good that is offered by a large number of vendors on a highly competitive market and for which an individual negotiation would come at prohibitive costs.

Therefore, the practical integration of negotiation mechanisms won't be feasible unless negotiation support is designed as an integral part of the overall software system. In the following, we will discuss the requirements for *automating* the negotiation process in the domain of E-Commerce and describe a system architecture which is currently being developed to provide the corresponding negotiation support for participants in electronic markets.

However, before starting to develop automated negotiation mechanisms, it is important to realize the scope and to recognize the potential advantages and disadvantages of such an automatism, since it can differ a lot from a conventional negotiation with human participants: First, in order to achieve a meaningful course, an automated negotiation usually needs to have an explicit goal which is pursued by a process formally specified by means of a protocol. Second, it can be assumed that there are no psychological aspects — for instance, there are no accompanying arguments — in the automated process, which can play a very important role in the conventional, human case. Third, due to the use of electronic media, a whole bunch of unforeseen chances as well as risks emerges in the automated case, for example, there are principally unlimited participation possibilities for a person or business party, on whose behalf a negotiation can be carried out. Generally, advantages of automated negotiations are performance (computing vs. thinking) and the potential to find more options or possible solutions in case of complex negotiation problems. Disadvantages are the potential predictability and exploitability due to the computability of the underlying algorithms and the possibility of failure due to the absence of psychological arguments (which can often be more convincing than formal ones).

In order to automate the task of negotiation completely, several of the automation mechanisms listed above have to be fully specified, developed and provided in an *integrated* manner: a communication language to exchange negotiation messages, a negotiation protocol specifying the possible courses of the negotiation process, and for each participant, a negotiation strategy to compute the negotiation action to be taken at a given time has to be provided. Then, a mechanism of "glueing" these functions together and integrating them into a component providing a uniform external interface to other participants is required. The most natural and flexible way to achieve this is apparently employing *agent technology*, since this programming paradigm provides exactly the right abstraction level: Each agent is an *autonomous* software component acting and negotiating on behalf of a person or business party. Especially, the deployment of *mobile agent* technology, which is being established the field of E-Commerce, offers additional advantages like location independence, asynchronous (mobile) user support, reduction of network traffic etc. However, since on the one hand the number and complexity of possible negotiation protocols and strategies is principally unlimited and the size of a mobile agent on the other hand should obviously be kept as small as possible, the incorporation of automated negotiation capabilities — as well as “intelligent” (reasoning) capabilities in general - into mobile agents could pose serious performance problems. Therefore, we have proposed a modular and very dynamic framework, in which the required negotiation capabilities can be subsequently loaded into a mobile agent or can be exchanged at run-time and which also allows explicit control of the mobility of an agent’s components. The design of corresponding agent’s modules providing negotiation capabilities including communication, protocol and strategy has been presented and the implementation of a plug-in mechanism to link such modules and embed them into mobile agents dynamically has been described.[14] In the rest of this section, we will focus on several *support services* to assist users to assemble their own negotiation enabled agents, especially to get their protocol and strategy modules, and to enable the agents to initialize and execute the automated negotiation in an efficient and reliable way.

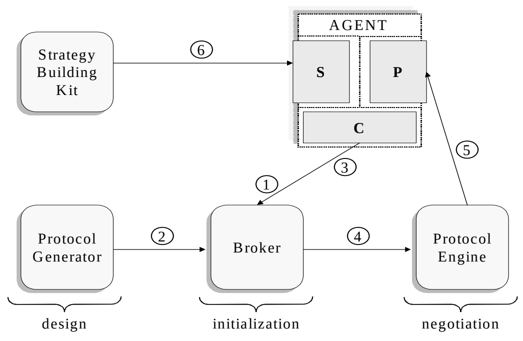

Fig. 2. Structure of negotiation enabled agents and support services.

Negotiation support services can be categorized as follows: At design time, services are needed to support users in designing and developing negotiation protocols and strategies. These services are called *protocol generator* and *strategy building kit* respectively and can be offered by third-party providers who also implement ready-to-use protocol and strategy modules from which a user can choose to equip his agent for a negotiation. In the initialization phase, a brokerage service (called *broker*) is needed to support the agents in finding appropriate negotiation parties. In the negotiation phase, an execution component, called *protocol engine*, can be employed to coordinate and monitor the interactions between the agents, thus making them behave conformable to the corresponding negotiation protocol.

- *Broker.* The functionality of this service has been described in section . The basic schema of the initialization of an automated negotiation using agents, in which the broker plays a central role, includes the following steps (as illustrated in Fig. 2):
  (1) An agent contacts the broker in order to initiate a negotiation on a certain issue.
  (2) Thereupon, he is required to choose a protocol template, which has been created and stored with the protocol generator, and register for a role in the template, e.g., for the role of the auctioneer.
  (3) Further agents register as interested parties for appropriate roles which have not reached the maximal number of instances, e.g. for the role of an auction visitor.
  (4) When the start conditions of the negotiation, for instance when the minimal number of visitors for an auction is reached, are fulfilled, all participants are notified and the completed template is passed to the protocol engine.
  (5) For each participating agent, the protocol engine generates a corresponding protocol module which is dynamically embedded into the agent frame.
  (6) Then, a proper strategy module fitting to the protocol module is chosen from the strategy building kit and plugged into the agent, and the negotiation process can now start.
- *Protocol generator.* Formal protocols to coordinate and monitor completely automated negotiations need to have an expressiveness which is strong enough to cover numerous requirements with respect to the aspects *issue of negotiation* (number, negotiable attributes, range of values), *participants* (roles, quantity, admission and exclusion conditions), *proceeding* (round specification and round number, voting method, timeout handling, truncation conditions), *validation* (syntactical and semantical correctness of offers and negotiation actions), and *bindingness* (scope of contractual binding of negotiated result to participants, bindingness of protocol itself). It is has been argued that static mathematical models are not appropriate to completely automate the negotiation process in an open environment such as the domain of electronic commerce, since they are not expressive enough to capture the dynamic aspects of such a process.[14] Indeed, a formally specified negotiation process can be seen as a specific type of *workflow* and therefore, the use of a colored Petri-net based language called OOPAMELA is proposed. Principally, OOPAMELA is a description language for colored Petri-nets enhanced by negotiation specific concepts including multiple roles, validation (of offers), voting method, round and timeout handling. In order to facilitate the development and usage of negotiation protocols expressed in this language, we have developed a protocol generator which provides support for graphical specification and parameterization of protocol templates and which can also be used as a persistent repository of ready-to-use "standardized" protocols.

  For example, the graphical specification of an auction protocol template using the current prototype of the protocol generator is shown in Fig. 3. Using the menu "Configurator" of this user interface, a wizard can be started which allows parameterization of this template like the minimal and maximal number of auction participants, validation condition for offers submitted and timeout for the transition "makeOffer".

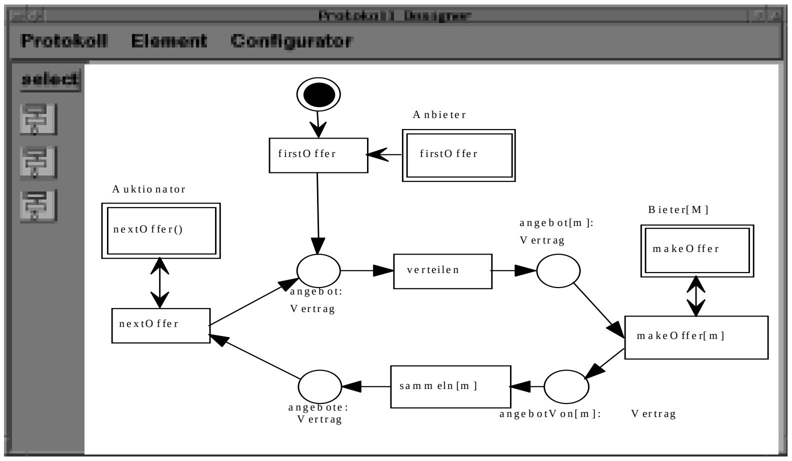

Fig. 3. Graphical Specification of an Auction Protocol using the Protocol Generator UI.

- *Strategy building kit.* Regarding negotiation strategies, it can easily be recognized that they can be based on very different algorithms, ranging from *analytical* to *evolutionary* approaches, or from *local* to *distributed* ones, which have all been proposed for some specific negotiation problems.[14,15] Moreover, these algorithms can normally be combined with different types of *knowledge bases* which can be exploited to produce a learning effect on the computation of negotiation actions, e.g., offers and counter-offers, the quality of which can be measured by some combination of utility functions such as price and quality (of products), time (to negotiate) etc. Usually, a good negotiation strategy makes use of both general *domain knowledge* like market values or branch statistics and *specific knowledge* like information about concrete negotiators, for example the result of the last negotiation with the same negotiator on the same issue. Due to this great variety of potential strategies that can be deployed for a concrete negotiation, a corresponding support service has to provide a *generic* and *dynamic framework* which allows for the use and exchange of strategy implementations based on different programming paradigms. Currently, in the context of the strategy building kit, we are developing a hierarchical, event-oriented framework in order to satisfy these requirements.

  In this framework (see Fig. 4), a strategy consists of a *coordinator* and one or more

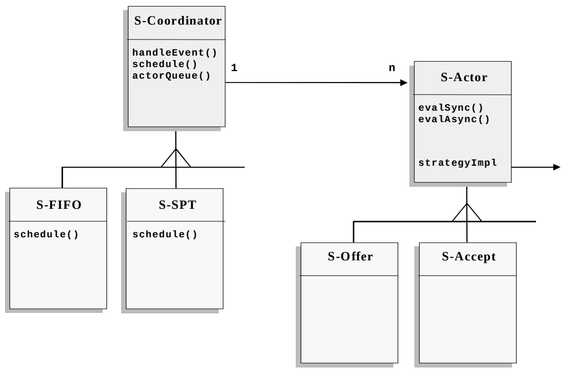

Fig. 4. A hierarchical strategy consisting of a coordinator and several concurrent actors.

actors. The coordinator is responsible for the handling of all external events, concerning for example new offers submitted by other participants or entry and exit of participants, and the scheduling of the actors. Each actor provides the implementation of a specific algorithm to compute negotiation actions like offering a new contract template or accepting and existing one. Several actors can run concurrently computing alternative actions which are then evaluated by the coordinator and during the computation, any or all actors can be interrupted by the coordinator as a consequence of an external event like exceeding the timeout to provide a counter-offer.

- *Protocol engine*. Basically, the task of the protocol engine is interpreting a negotiation protocol which has been generated and completely parameterized (using the protocol generator) and "supervising" the participants in such a way that the semantics of the protocol is guaranteed. The interface between the protocol engine and the participating agents is the *protocol module*. This is a component generated by the engine which can be dynamically plugged into the agent frame and provides methods to inspect the contents of incoming and outgoing (relatively to the viewpoint of the respective agent) negotiation messages. The protocol engine, considered as a workflow engine (see also Section "Contract Execution" below), has to be able to deal with many exceptional situations which can result from incorrectly specified protocols (e.g., deadlocks), but which can also be (purposefully) caused by the participants (e.g., trials to deceive, hinder or bypass the protocol module).

The described mechanisms to enforce protocols assure safety and security in an automated manner. But since business is always a critical thing in humans´ minds, a realistic infrastruture should provide more "traditional" means of assurance as well — signatures.

5. Electronic Contract Signing

From the legal perspective, contracts don't need to be signed nor even be written. They become valid even if they are closed orally or by a conclusive deed. I.e., when a customer hits the "Buy" button of an electronic shop application, it can be assumed that the consequences are well-known. On the other hand, there are several reasons that promote the idea of involving signed electronic contracts into online transactions:

- A written contract *cannot be repudiated*. In the case of an electronic contract, this can be signed by the parties as well as by a trusted third party. This states who agreed on which terms and at which time. Any arbitration that may be required among these parties can be settled better if there is a version of the contract available that has been archived by a neutral auditor.
- The *legal framework* for online commercial transactions is being established in several countries now. For example, in Germany, the Regulatory Authority for Telecommunications and Posts (RegTP, "Regulierungsbehörde") is responsible for the licensing of service providers in the field of electronic signatures according to the Signaturgesetz (Signature Act). Under http://www.regtp.de the directive on digital signatures can be checked in German and in English as well as a list on technical requirements of the digital signatures. As far as electronic contracting is concerned, electronic signatures are thus at least accepted as an authentication means for the document signed. However, the management of a contract still requires a further harmonization of the national legislation for the participating countries.
- *Complex legal situations* can be better fixed by using a document as the common form of agreement. Today, it is best practice that commercial vendors display their terms and conditions as a part of their online presentation. However, it would clarify the legal situation if these documents are not displayed transitionally on the Internet but if they could be escrowed and archived at a third party (e.g. a notary). This would allow the contracting parties to refer to this document even a long time after it has been replaced by a new one.
- Furthermore, some contracts may be negotiated and closed that require *too complex specifications* such that it is essential to handle them in written form as a shared document. This applies to project workplans as well as to complex relationships for obligations and rights within multilateral agreements.
- In contrast to paper-based contracts, their electronic counterparts could be *executable*. Structurally, such contracts incorporate clauses that determine the obligations and rights of each party. From the technical point of view, this can be interpreted for many contracts as an activity or a service that is to be provided at a certain time (payments, delivery of a good or a report, translating a document, or printing, binding and delivering books). Therefore, the *execution phase* of the commercial transaction is not only interpreted in the legal sense as the execution of a contract, but specifically in a technical sense by invoking the corresponding service through remote method invocations.

6. Electronic Contract Execution

Up to now the monitoring and the execution of the agreed terms in a contract are, in many cases, a time consuming and complex task that probably bears one of the highest potentials in cutting costs. However, electronic contract execution is not available in current E-commerce infrastrucutres. These are reasons why the COSMOS system does not only support the creation, negotiation and signing of contracts, but also their automatic execution.[19] The following subsections briefly depict some of the key aspects of the on-going COSMOS project that aims at establishing such a complete infrastructure.[20]

6.1. The COSMOS Contract Model

In the common case of a printed contract, the document contains all the information about its execution in a format that is usually not machine-readable. The duties and rights of the parties, the schedule of the performance and the identities of the involved parties: all this currently needs to be extracted and interpreted by humans. In many cases, even experts are needed to evaluate a contract.

Basis for an automatic contract execution is its machine readability. Using an electronic contract format offers the chance to create a contract model that may be easily interpreted by computers to supervise its fulfilment.

The COSMOS architecture has been considering this from the very beginning. As a part of the system, a contract model has been developed that can be used to represent most types of contracts between n parties.[7] This model allows an easy extraction of the contract's *workflow information*. Consequently, the information gathered within the contract during earlier phases directly allows to derive a workflow to execute a contract.

For a better understanding of how workflow information is included in a contract, we depict our contract model: from an abstract point of view the workflow information can be found in the three main components (see Fig. 5) of a contract:

- The *Who*-part defines the involved parties, their roles and the persons representing the parties.
- The *How*-part describes the execution steps of the contract by a number of activities that have to be accomplished.
- The *What*-part contains the information about the contract's subject. It defines the obligations to be brought forth.

Taking a closer look, these contract components can be seen under the following relationships:

Each *activity* defines one step of the contract execution and refers to two roles: one represents the group of the receivers of a number of *obligations* and the other characterizes the group of parties having to perform these obligations. Each *party* embodies one or more persons, the people who signed the contract and who are responsible for its fulfilment. Furthermore, the person objects can contain information on how to contact the person or his workflow system, so the defined activities may be initiated.

This object-oriented contract model allows to define different attributes of actions like the time, pre- or postconditions of single execution steps. Moreover, actions may relate to each other, e.g. the transfer of a payment may only be initiated after a good has arrived.

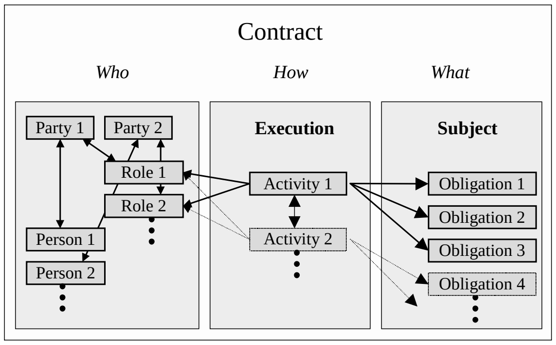

Fig. 5. Main Components of an Electronic Contract.

6.2. The COSMOS workflow engine (CWE)

The component of the system responsible for the contract execution support is the COSMOS Workflow Engine (CWE). It consists of several parts and provides three different contract execution models.

The first task of the CWE is to gather the workflow information from a signed contract. A Petri-net like representation is used to represent the workflow information inside the CWE (see also section).

In the next step, the system relates each activity to the kind of contract execution support the particular user prefers. This information is defined in the *person* object. Also persons registered in the *profile database* may store information on their business applications in *business object adapters*.[21] Here everyone can define an own business object adapter that gives the CWE the possibility to communicate with the user's business application.[22]

To help users to define a specific contract execution, the system supports them by offering specific contract templates. It is also possible to define several elements of the contract execution within an offer already.

6.3. Models of contract execution

The COSMOS infrastructure provides three different models of electronic contract execution (see Fig. 6). They differ in the degree of automation. The appropriate model is depending on the contract's nature and the given infrastructure of the parties involved in the contract.

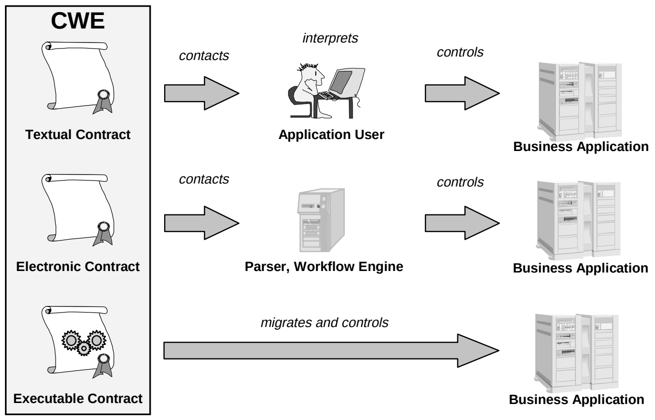

Fig. 6. Three Models of contract execution.

- In the elementary case, deadlines and periods of contract obligations may be extracted by the CWE to generate e-mail notifications. These messages refer to the actions the parties agreed on, e.g. initiating a payment or performing an action. The e-mail itself contains all necessary information the user needs. If appropriate, the e-mail may contain a link to an applet that shows the contract's execution status, or which helps the user to fulfil his task. For example if a payment has to be done, a link to the home banking application may be provided. We call this the *supportive* workflow system model.
- At a more sophisticated level, the CWE interprets the contract on a central server which can directly contact the workflow systems of the participating parties. In this case, the CWE is an abstraction of an usual workflow system which has adapters to different workflow product environments. Single activities triggered by the workflow engine are mapped to data messages or method calls of the participating business objects (BOs) extracted from the contract. Since these BOs were originally issued by companies offering their services to the catalogue service, each BO "knows" its interface to an individual company's environment. Thus, the contract components are able to directly contact and drive a company's workflow system in a consistent and correct manner. For example, a necessary payment in the workflow would cause an invocation of the company's financial management software, fill out an "electronic form" and initiate the payment process. For the communication between the CWE and different workflow system types, an XML, as well as an EDI and a CORBA-interface, are under development.[20] In this scenario, the workflow control is centralized at the CWE, therefore we call this approach the *integrative automated* approach.
- The third model attempts to support the contract execution on a distributed basis. In this case, the workflow is not controlled by the CWE directly, but

the CWE generates a number of agents which are migrated to the participants´ computers where they control the fulfilment of the contract independently. These agents can be considered as an active version of the contract. They contain the methods and the information necessary to perform certain actions of the contract workflow, but usually no textual paragraphs are attached.
Several specialized agents may be created, which migrate to different contract partners. These programes can communicate with each other to control the correct order of the actions. After having carried out one action, the agent may even migrate to an other workflow system assuring a specific order if necessary. At this stage, the centralized CWE is no longer a full-fledged service, why we call this scenario the *distributed* workflow system approach.

Each of these approaches bears different potentials and problems. We are evaluating all three models in practice, in order to fit the needs of a wide range of users and to explore new system techniques.

Especially the third approach seems to bear several advantages (although it is the most sophisticated), as the interactions necessary for a contract execution are highly distributed by nature. This approach requires the ability of objects to migrate over the network as well as some infrastructure that can receive and store these migrated objects, but it avoids the client-server bottleneck of the integrative approach.

The distributed model is based on Object-Space's Voyager technology[23] and Sun's JavaSpaces.[24] Voyager is used for the migration of objects as well as for remote communication and messaging. JavaSpaces serves as an asynchronous message buffer and storage mechanism in a distributed environment. Participating servers need to provide a Voyager demon and JavaSpaces both being software with a very small footprint. But companies do not need to have established their own internal workflow system. They simply provide the business objects that should be deployed during a contract´s execution.

Still, with all three approaches humans may have (the wish) to *read* the contract. Although the COSMOS system uses one contract model for all kinds of contracts, different types of presentations may be selected. This is achieved by an XML-representation of the contract´s object model and different (eXtensible Style Language) XSL-Definitions which are used to display or print a contract.

For the integrative automated and the distributed model, security is a big issue. Avoidance of abusing the provided interfaces and keeping unauthorised agents away is assured by every company using the system being able to create their own business object adapters and save them with their profile information in the cosmos profile database acting as a trusted third party.

We also provide a component model that helps the users to create a contract from a set of pre-defined components. The following section reflects on the influence of the recent component-oriented software discussion to our infrastructare in general.

7. Component-Software

Services offered in an open and heterogeneous environment such as an Internet-based electronic contracting service can only be competitive if they are flexible enough to adapt to a wide variety of user as well as technical requirements. Therefore, appropriate techniques are needed to (re-) configure an offered service dynamically according to the effective requirements in each case. For example, regarding the involvement of so-called "third party" services — such as payment and notary functions — during a business-to-business transaction, as many options as possible should be supported and moreover, an option common to all transaction parties has to be determined and activated.

A very generic approach to provide system support for such kind of dynamic service configuration consists of using *policy management* mechanisms, which provide a formalization of arbitrary requirements in terms of policies which can be evaluated, compared, matched (or unified) and activated in an application independent way.[16] Policies can be added and activated fully automatically at run-time without changing the application code. The configuration effect is achieved by modifying externally accessible properties which are used as system parameters by the applications.

In a broader sense, flexible service configuration also means that arbitrary services should be so configurable that they can be easily plugged together to yield new functionality, i.e. that they can be used as building blocks to assemble new services "on the fly". Providing technical support to fulfill this kind of requirements is precisely the objective of componentware techniques which seem to be the right means to face the growing challenges in the field of electronic commerce, especially concerning the requirement of dynamic adaptability.[17] However, there are still a lot of open questions and unsolved problems — for example, the *composition* of an application system out of components has to be distinguished from the *generation* of a new component out of existing ones since they have to fulfill, among other things, very different performance requirements — which are currently being investigated in projects such as DynamiCS in order to make componentware technology applicable in practice.[18]

The notion of *componentware*[17,30] as a means to describe software systems consisting of loosly coupled modules with clearly defined interfaces has a variety of aspects that makes it worth to be considered as an architectural as well as a conceptual base for a complex E-commerce infrastruture.

Usually, *reuse* is emphasized as the main advantage of a component-oriented view on software.[27,28] But this *still* has to be proven in everyday´s application programming and we will not discuss this issue here in any detail. Nonetheless, the COSMOS system as an electronic contracting platform has several points at which meaningful and successful reuse of existing components occurs quite naturally. For example, the first couple of sections above demonstrated the deployment of broker mechanisms as well as that of the so-called contract editor within different phases of the construction and fulfilment of a contract. Here, the tools and service components are (re-)used in different situations at some time with at other times even without human interaction.

Aside from reuse, the *strutural* aspects of componentware seem to be promising for a large scale software system with different and changing needs at different places. Decomposing — or to say it the bottom-up way, building up — a system´s architecture into separated components allows better and more suitable design of the overall system, particularly in the case of a highly distributed system, since these are loosely coupled in a natural way. But again, this is a long and well treated topic in literature not being considered in-depth here.[25,26]

We are not going to discuss methodological aspect of applying component-based software either.[29] Rather, we like to stress the multiple levels of abstraction components occur within our contracting infrastructure. First, there clearly are the "building blocks" like the broker, the contract editor, or the workflow engine. Second, there are components identifiable at a more fine-grained level, in particular the protocol and strategy modules for the negotiation suppport (see section). The third — and from a certain point of view most important — level of components are the aforementioned *business objects* that our architecture deploys for service profiling, within contract templates and during the workflow-like execution of a contract.

In fact, the notion of business objects[21] — usually defined as "the objects being of importance to a business (process)" — can be seen as an intermediate step between the concepts of object–orientation and component-software. This holds particularly if one stresses their encapsulating and reificating character making them "black-box" units of deployment and representation within a business.

Section describes the features of a catalogue service as it is deployed by a system infrastructure like COSMOS. The concept of a catalogue includes the notion of relating one (searchable) entry to another one. As a result, a simple name-value pair representation will be sufficient for most catalogues. This provides the addtional advantage of simplicity (of implementation). But the description of advanced service mediation (section) and service execution (section) should have made clear that more information is needed for such highly automated mechanisms.

Therefore, we adopt the notion of business object adaptors from OMG´s BOCA[21] to abstract from simple name-value pairs to components comprising full business objects, including code for their later usage during contract fulfilment, for example. Using adaptors avoids to force the system infrastructure or its users (companies) to deploy full support for (distributed) object-orientation — OODBs for example — and the handling of complex technologies, but still offers the opportunity to do so — particularly in the future. We have described the details of this adaptation.[22]

We like to conclude by pointing out what we think is the most important issue of componentware in the context of a large scale E-commerce infrastructure: to summarize it is all about *evolution*, *longevity*, and *maintainability*. In some of the advanced stages of a project like COSMOS when we had to decide what *technologies* (e.g., communication layer, programming language etc.) to deploy the discussion arose if *whatever* technology we elect — would it survive long enough to support contracting scenarios that even may run for years (think of global business-to-business settlements). The outcome of this discussion was the demand for an as flexible, configurable and customizable system architecture as possible making it an evolutionary infrastructure.

This is especially important, since E-commerce should not repeat the old mistake of businesses basing their key processes on the deployment of monolithic, ever-growing software systems that hardly can be maintained. The world-wide emerging E-business has to avoid itself needing to be "re-engineered" in future years.

Therefore, our technologies our accompanied by supporting services like traders, policy- and type-managers that allow for customization and inspection of meta-level information even at a system´s run-time. Also, upgrading and substitution of single parts in the system becomes possible.[17] The notion of software components is highly advantageous here, because a component is seen as a pre-configured but still customizable piece of software: the component developer has included precisely defined *variation points* a later user (or application level programmer) may customize to her needs without the danger of inconsistencies or failure of the system as a whole. By giving automated mechanisms access to this customization we end up with a self-controlled system assuring the maintainability, stability and robustness as one expects (or demands) of a business-critical infrastruture.

8. Conclusion and Outlook

In this paper, we have presented an overview of system technology concepts that will allow and ease the establishment of complex electronic commerce infrastructures. In particular, we outlined aspects of a system capable of supporting electronic contracting including all phases from information gathering (finding business partners), negotiation (coming to terms with the partners) and the fulfilment of the resulting obligations (contract performance).

Our aim was to demonstrate the possibility of harmonic combination of simple, well-known existing technologies with advanced concepts and system and application level services being still under research. We strongly believe that such a combination added to today´s open distributed networked computing environments will allow even small and medium companies (or even single persons) to participate in tomorrows global electronic market without being hindered by high set-up costs, the demand for complex infrastructures or the need of detailed technical, economical or legal (possibly international) knowledge.

To achieve such a complete architecture, we have to consider features of quite a couple of different information technology areas, ranging from type theory, distribution, negotiaion, user interface design to componentware. We have gained a lot of experience in these areas during different projects we briefly introduced in this article. There exists a good number of proof-of-concept prototypes of all the mentioned techniques right now. Nonetheless, our main duty in future will be the smoothless integration of all these parts into a sound and applicable system being real "commercial of the shelf" (COTS) software. We consider this final step (and of course its field tests under real-world conditions) as essential for an architecture that claims to support *business*.

Acknowledgements

The projects that led to the work presented here are funded by the European Commission (Esprit Project #26850) and the German Research Council (DFG Project La1061/1–1).

References

1. Z. Milosevic, Enterprise aspects of open distributed systems, PhD Thesis, University of Queensland, Australia, October 1995. `www.dstc.edu.au/AU/staff/zoran-milosevic.html`
2. Electronic Commerce Homepage of the European Commission: `http://www.ispo.cec.be/ecommerce`
3. B. F. Schmid and M. A. Lindemann (eds.), *Proceedings of the 31st Annual Hawaii International Conference on Systems Science, HICCS'98*, Hawaii, January 1998, **IV** (19998) 193-201.
4. ISO/IEC, ITU-T Rec. X.950 — ISO/IEC 13235–1 — ODP Trading Function — Specification, International Standard, International Standards Organisation, ISO/IEC JTC1/SC21, December 1997.
5. AT&T, DSTC, DEC, HP, ICL, Nortel, and Novell, Trading Object Service, OMG Document No.: orbos/96-05-06, Version 1.0, 1996.
6. Onyx Internet Ltd. Tradezone, `http://www.tradezone.onyx.net`
7. M. Merz, B. Liberman, E. Wolff, M. Boger and H. Weinreich, COSMOS Deliverable D2 – Reference Architecture, *Project Deliverable*, (University of Hamburg and Ponton Hamburg, 1998)
8. P.A. Bernstein, Middleware – An Architecture for Distributed System Services, CRL 93/6, Digital Equipment Corporation, Cambridge Research Lab, 1993.
9. L. Cardelli and P. Wegner, On Understanding Types, Data Abstraction, and Polymorphism, in ACM Computing Surveys no.4, vol.17, 471–522, Dec. 1985.
10. K. Mueller–Jones, M. Merz and W. Lamersdorf, The TRADEr: Integrating Trading Into DCE, in *Proceedings of the 3rd IFIP TC6 Conference on Open Distributed Processing, ICODP'95*, eds. K. Raymond and L. Amstrong, Chapman & Hall, Brisbane, Australia, 1995.
11. K. Mueller–Jones, Koordinierte Dienstnutzung in offenen verteilten Dienstemaerkten, PhD Thesis, University Of Hamburg, September 1996.
12. S. Mueller, K. Mueller-Jones, W. Lamersdorf and T. Tu, Global trader cooperation in open service markets, in *Proc. Workshop Trends in Distributed Systems: CORBA and Beyond*, eds. O. Spaniol, C. Linhoff-Popien, B. Meyer, Lecture Notes in Computer Science **1161** (Springer, Heidelberg, 1996) 214–227.
13. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, G. Wiederhold, An overview of KQML: A knowledge query and manipulation language, Technical Report, April 1992.
14. M. T. Tu, F. Griffel, M. Merz, and W. Lamersdorf, A plug-in architecture providing dynamic negotiation capabilities for mobile agents, in *Proc. 2nd International Workshop on Mobile Agents, MA'98*, Stuttgart, eds. K. Rothermel and F. Hohl, Lecture Notes in Computer Science (Springer, Berlin, Heidelberg, New York, 1998).
15. C. Beam and A. Segev, Automated negotiations: A survey of the state of the art, Negotiation and Collaboration in Electronic Commerce Project Document 96-WP-1022, Berkeley, 1997. `http://haas.berkeley.edu/~citm/nego-proj.html`

16. M. T. Tu, F. Griffel, M. Merz, and W. Lamersdorf, Generic policy management for open service markets, in *Proc. IFIP International Working Conference on Distributed Applications and Interoperable Systemes, DAIS'97*, eds. H. Koenig and K. Geihs (Chapman & Hall, London, Weinheim, New York, 1997) 211–222.
17. F. Griffel, *Componentware* (dpunkt-Verlag, Heidelberg, 1998).
18. DynamiCS Project Home Page:
`http://vsys-www.informatik.uni-hamburg.de/projects/dynamics/index.phtml`
19. F. Griffel, T. Tu, M. Muenke, M. Merz, W. Lamersdorf, and M. Mira da Silva, Electronic contract negotiation as an application niche for mobile agents, in *Proc. 1st International Workshop on Enterprise Distributed Object Computing*, October 1997.
20. COSMOS Project Home Page: `http://www.ponton-hamburg.de/cosmos`
21. Object Management Group, CORBA BOCA – Business Object Component Architecture, Specification, OMG Document Nr. bom/98-01-07, 1998.
22. M. Merz, F. Griffel, S. Mueller-Wilken, and W. Lamersdorf, Electronic contracting with COSMOS — how to establish, negotiate, and execute electronic contracts on the internet, in *Proc. 2nd International Workshop on Enterprise Distributed Object Computing*, San Diego, Nov. 1998.
23. ObjectSpace, Voyager — Core Technology User Guide, 1997
`http://www.objectspace.com/voyager/documentation.html`
24. `http://chatsubo.javasoft.com/javaspaces`, 1998.
25. D. Krieger and R. M. Adler, The emergence of distributed component platforms, IEEE Computer **3** (1998) 43–53.
26. D. L. Parnas, On the criteria to be used in decomposing systems into modules, Communications of the ACM **15** (1972) 1053–1058.
27. J. Sametinger, *Software Engineering with Reusable Components*, (Springer, 1997).
28. I. Jacobson, M. Griss, and P. Jonsson, *SOFTWARE REUSE — Architecture, Process and Organization for Business Success*, (ACM Press / Addison–Wesley, 1997).
29. P. Allen and S. Frost, *Component-Based Development for Enterprise Systems*, (Cambridge Univ. Press, New York, 1998).
30. C. Szyperski, *Component Software — Beyond Object–Oriented Programming*, (Addison–Wesley, 1998).
31. S. Adler, W. Lamersdorf, M. Muenke, S. Ruecker, H. Spahn, U. Berger, A. Brueggemann-Klein, and C. Haber, Grey literature and multiple collections in NCSTRL, in *Digital Libraries in Computer Science: The MeDoc Approach*, eds. A. Barth, M. Breu,A. Endres, A. de Kemp, Lecture Notes in Computer Science **1392** (Springer-Verlag, Berlin Heidelberg, New York, 1998) 45–170.
32. Commerce Net Home Page: `http://www.commerce.net`
33. F. Griffel, T. Tu, W. Lamersdorf (eds.), *Electronic Commerce* (dpunkt-Verlag, Heidelberg, 1998).
34. `http://www.ibm.com/Java/Sanfrancisco/technical.html`, 1998.
35. W. Lamersdorf and M. Merz (eds.), *Trends in Distributed Systems for Electronic Commerce*, Lecture Notes in Computer Science **1402** (Springer-Verlag, Berlin, Heidelberg, New York, 1998).
36. S. Mueller, K. Mueller-Jones, W. Lamersdorf and T. Tu, Global trader cooperation in open service markets, in *Proc. Workshop Trends in Distributed Systems: CORBA and Beyond*, eds. O. Spaniol, C. Linhoff-Popien, B. Meyer, Lecture Notes in Computer Science **1161** (Springer, Heidelberg, 1996) 214–227.
37. S. McConnell, M. Merz, L. Maesano, and M. Witthaut, An open architecture for electronic commerce, OMG/ECDTF/OSM Response, 1997.