Trading Services in Ontology-driven Markets

Steffen Lamparter
Institute AIFB
University of Karlsruhe (TH)
Germany
sla@aifb.uni-karlsruhe.de

Björn Schnizler
Information Management and Systems
University of Karlsruhe (TH)
Germany
schnizler@iw.uni-karlsruhe.de

April 30, 2005

Abstract

In recent years, Web Services have become the key technology for building flexible and interoperable computing infrastructure. However, to realize the vision of a full-fletched service oriented architecture efficient service discovery and allocation is required to coordinate the interplay between service providers and requesters. Recently, the Semantic Web community suggested the use of Semantic Web Services and semantic matchmaking mechanisms for allocating adequate services. These mechanisms do not adequately address or even neglect the existence of prices for services and thus may lead to economic inefficient outcomes. Cleaving on the advantages of a semantics based matchmaker, this paper describes the design of an ontology-driven market for trading Web Services. As such, a framework is presented that enriches an auction schema by several components enabling semantics based matching as well as price-based allocations of services. Furthermore, the use of background knowledge in auction mechanisms also reduces the overall complexity of the auction which is shown by means of a simulation.

1 Introduction

Web Services have become the key technology for enabling distributed computing infrastructures for collaboration and electronic commerce. They can be used to automate business processes, to enable process-driven application integration, and to deploy Computational Grid infrastructures. The benefits of using Web Services are improved

connectivity, efficiency, flexibility, immediacy, and interoperability of disparate systems [1]. Although in theory, users could utilize Web Services from multiple locations, this potential is rarely exploited in practice. This seeming contradiction is caused by the significant barriers that arise when organizational boundaries are crossed. Overcoming these barriers requires that Web Services can be reliably discovered, acquired, and managed.

Most of the current service discovery mechanisms for Web Services use matchmaking algorithms which rely on attribute-based matching functions [2]. These algorithms are capable of syntactically comparing user requirements against service descriptions. However, a symmetric and attribute-based matching of service descriptions is inflexible and difficult to extend to new characteristics or concepts [3].

Recently, the Semantic Web community suggested the use of Semantic Web Services (e.g. using OWL-S [1], WSMO[2]) which enrich services by a formal and explicit description of their capabilities. Instead of syntactically based matchmaking algorithms, services can be semantically matched using concepts and relations formalized by means of ontologies [4]. Furthermore, these matchmakers can be easily extended by adding new concepts and rules and adapt the service discovery to changing policies.

However, the direct application of matchmaking mechanisms for allocating Web Services has several drawbacks: Firstly, these algorithms do not guarantee that those requesters will receive the supplied services who value them most. Secondly, they ignore the fact that users will only offer their services if they are adequately compensated. Compensation requires determining how the offered services are allocated among potential requesters and how the prices for the services are set. Thirdly, semantics based matchmaking algorithms are typically computational demanding and thus not adequate for large-scaled negotiations. However, these aspects are crucial for implementing economic efficient Web Service infrastructures.

Recently, researchers have suggested employing market mechanisms for the allocation of Web Services [5, 6]. Markets can be an effective institution to allocate resources (Pareto-) optimally [7]. This is achieved by the interplay of demand and supply and due to the information feedback inherent to the price system. As such, the application of market mechanisms for the service discovery is deemed promising.

Applying classical market mechanisms[3] for trading Web Services may lead to inefficient outcomes as these mechanisms allocate on base of syntactical descriptions. The extension of semantic matchmakers to auctions is, however, from a computational point of view intractable. Rather, existing market algorithms which perform well have to be enriched by components capable of handling semantically represented service descriptions.

The contribution of this paper is twofold: Firstly, a market platform for trading Web Services efficiently is presented. This is realized by merging the advantages of classical auction algorithms and semantically based matchmakers into one framework. Secondly, it is shown that the complexity of a classical auction mechanism is reduced by means of formalized background knowledge.

[1] `http://www.daml.org/services/owl-s/1.1/`

[2] `http://www.wsmo.org/`

[3] For instance, the mechanism used in eBay (`http://www.ebay.com/`).

The paper is structured as follows: In section 2, the design requirements for an ontology driven marketplace for trading Web Services are elicited. Based on these requirements, a market architecture is outlined in section 3. In section 4, an ontology based communication language is introduced for specifying service offers, requests, and agreements. Section 5 describes the basic components of the marketplace infrastructure. Section 6 evaluates the splitting algorithm by means of a performance simulation. Section 7 compares the marketplace with related work and section 8 concludes the paper.

2 Design Requirements

The design of a market platform for trading services mainly affects two components [7]: (i) *A communication language* which defines how bids (i.e. offers and requests) and agreements can be formalized and submitted to the market mechanism. (ii) *An outcome determination* by means of an allocation (i.e. who gets which service and a pricing schema) and a payment component. Both components as well as the interplay between them have to be designed carefully to reflect the users' requirements [7]. Besides classical properties stemming from the mechanism design theory (e.g. allocative efficiency, incentive compatibility), the following domain specific requirements for a market platform capable of trading services can be identified [8, 9]:

R1 *Immediate execution* of possible counterparts is required by the mechanism. This bases on the fact that services are usually time critical goods.

R2 *Simultaneous trading* of multiple sellers and multiple buyers is required by the mechanism, i.e. sellers and buyers have to be allowed to place bids simultaneously.

R3 *Trading heterogenous services* simultaneously is required to be supported by the mechanism, e.g. computational services and storage services.

R4 *Meaningful matchmaking of orders* should be realized by the market infrastructure to allow a matching of services based on the semantics of an order instead of their syntactical representation. For instance, a request for a storage service should be matched with an offer for a hard disk service.

R5 *Trading dependent services* is required to fulfill the dependence property of services. For instance, buyers usually demand a combination of services as a bundle to perform a task, e.g. a computational service in combination with a storage service. This is based on the fact that services can be complementarities, i.e. the valuation for a set of services is greater than the sum of the valuation for each service. Thus, it is required to support bids on bundles[4]. Furthermore, a buyer may want to submit more than one bid on a bundle but many that are excluding each other. In this case, the services are substitutes, i.e. the buyer has subadditive valuations. As such, the market must support XOR-bids to express substitutes.

[4] A bundle can be defined as an AND concatenated bid on multiple services, e.g. $S1 \wedge S2$.

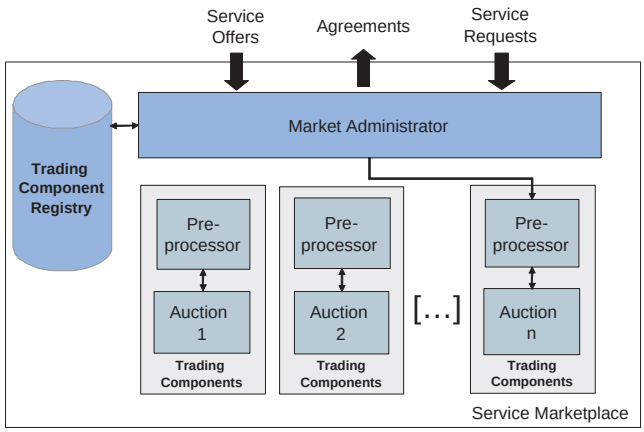

Figure 1: Overall architecture of the service marketplace

R6 *Support for multiattribute services* should be realized by the market as services are typically not completely standardized. Similar services can differ in their quality and their policies, e.g. a storage service by its capacity or access time; a billing service by its age restriction.

R7 *Computational and communication efficient determination of the outcome* is required by the mechanism in order to converge on a desirable global outcome by minimizing the computational effort.

Without loss of generality, we restrict the family of market mechanism to auctions, as auctions are an efficient institution for allocating services and determing prices [7].

3 Marketplace Architecture

In this section the overall architecture of an ontology-driven marketplace is introduced. The architecture, as outlined in figure 1, consists of the following components: a Marketplace Administrator, a Trading Component Registry, and several instances of Trading Components which encapsulate Preprocessors and Auction instances.

Although the market is seen as a monolithic unit, internally it may consist of several independent trading components each with its own auction mechanism and preprocessors. In each of these individual trading components, only a subset of all available services is traded. These sets are disjunctive subsets of all services and may include several dependent heterogenous services which are demanded or supplied in a bundle. For instance, a trading component TC_1 could include storage services and computational services because several requesters are requesting these services as a bundle. Another trading component TC_2 could only include payment services because users demanding or supplying this service are not interested in trading storage services as a bundle.

Although a central auction mechanism comprising all available services would fulfill the economic properties, a distinction between several independent trading components clearly reduces the complexity of the auction (R7), especially if a great number of heterogeneous services are traded.

Trading Component Registry The Component registry is a repository that contains an ontology-based description of each trading component. These descriptions basically specify the capability of the services that are traded in a specific trading component.

Marketplace Administrator The Marketplace Administrator manages the internal mechanisms, i.e. the administrator instantiates, removes, or merges trading components. Furthermore, the administrator receives orders from requesters and suppliers. In order to allow meaningful matchmaking (R5) these orders are formally expressed via ontologies. The Marketplace Administrator compares the capabilities specified in the order with those stored in the registry on a semantic level (R4) and forwards the order to the corresponding internal trading component.

Auction In order to compute the optimal outcome, a multiattribute combinatorial double auction is applied [8]. This mechanism meets the requirements mentioned in the last section by allowing immediate executions (R1), simultaneous trading (R2), and trading heterogeneous (R3), dependent (R4), as well as multiattributive (R6) services. However, existing implementations of such mechanisms rely only on pure syntactic matching of orders and, thus, requirement (R4) cannot be met. Therefore, an additional preprocessing of the order book is necessary.

Preprocessor The Preprocessor administrates an order book containing ontology based orders and is responsible for preparing this order book in a way that it can be handled by a traditional auction system. Thereby, two aspects can be distinguished: (i) *Semantic preprocessing:* By means of the taxonomic relations derived from the ontology bids are submitted or removed in a way that that the subsequent syntactic auction mechanism results in the same set of matches as a semantic approach, while inappropriate allocations are avoided (R4). For example, a storage service is requested and a hard disc service offered, syntactic matching will fail due to the lack of taxonomic information. Therefore, semantic preprocessing adds an additional bid, in this case a hard disc service request, which is XOR connected with the request for a storage service. Hence, the hard disc request can be matched with the corresponding offer while preserving the consistency of the order book. (ii) *Syntactic preprocessing:* Once this is done, the ontology based communication primitives are translated into the syntactic bidding language that is understood by the auction mechanism. Thereby, the ontology instances are serialized to strings.

Before the marketplace components are described in more detail, a language for specifying requests, offers, and agreements is introduced in the following section.

4 Ontology-based Communication Language

Formal knowledge representation is crucial for providing a marketplace that allows intelligent and meaningful allocations. Ontologies provide the right means by featuring logic-based representation languages [10]. They come with executable calculi that allow querying and reasoning during run-time. Besides, such formalisms facilitate the conceptual integration of heterogeneous service description efforts by providing well-defined and machine understandable semantics. Before the main communication primitives and ontological concepts are introduced in section 4.2, 4.3 and 4.4, major design considerations of the ontology are discussed in 4.1.

4.1 Ontology Design Considerations

Currently there are several different initiatives that strive for describing different aspects of Web Services semantically (e.g. using WSMO, OWL-S). In order to be able to handle the different external communication languages, the marketplace is based on the OWL-DL version of the foundational ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [11]. Foundational ontologies are high-quality formalizations of domain independent concepts and associations that contain a rich axiomatization of their vocabulary. They provide a philosophical sound reference point for easy and rigorous comparison and thus, can be used for harmonizing and integrating existing specifications like WSDL, OWL-S, WSMO, WS-Polciy, XACML, etc. This is done by aligning existing specifications with their upper level concepts derived from DOCLE (as done for OWL-S in [12]).

Descriptions & Situations (D&S) [13] is a module that extends DOLCE in order to allow modelling contexts such as social institutions, regulations or roles. By introducing the notion of *Descriptions*, DOLCE ground entities can be described from different points of view. For instance, in one context a Web Service might play the role of a requester and in another context that of a service provider. Further, such contexts can be used to distinguish between a service description and a service in the real world. In the following, offers, requests, and agreements are specified by means of *Descriptions* of services.

4.2 Communication Primitives

The interaction between participants and marketplace mainly affects the three communication primitives *Offers*, *Requests*, and *Agreements*. These are also the major concepts that represent information in a market. As depicted in figure 2, offer, request, and agreement descriptions can be regarded as a specialization of the concept *Service Description*, which is a subclass of the upper level concept *Description* derived from D&S (see also [14]). A *Service Description* specifies concrete service capabilities and constraints of a service, which are also modelled as *Descriptions* in terms of D&S. A *Service Description* must have at least one *Capability Description*, whereas constraints can be defined optionally. In the following the main communication primitives offers, requests, and agreements are discussed in more detail.

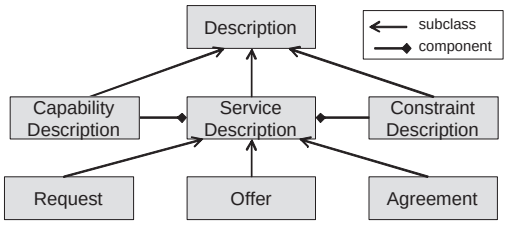

Figure 2: Relations between D&S descriptions

Offers: A service provider sends an offer to the marketplace which contains concrete information about the capabilities that will be carried out by the service. Furthermore, an offer contains a reservation price for the service. Thus, an offer can be regarded as a *Service Description* containing a mandatory constraint that defines an lower bound for the price a provider asks for. For instance, an offer for a stock quote service may contain information about the average response time, payment restrictions (e.g. only credit cards), and mandatorily a reservation price (e.g. at least 5$). Offering bundles of services is ontologically represented by a *Service Description* that contains more than one *Capability Description*[5].

Requests: A service request is the counterpart of an offer and specifies the requirements of a customer. A request contains information about the minimal requested capabilities of the service, e.g. a stock quote services that returns quotes for companies listed in the Dow Jones in Dollar. Furthermore, a request contains the valuation of the customer for the service, i.e. the maximum price a customer is willing to pay for the service, e.g. at most 6$. Thus, a request can be regarded as a *Service Description* containing a constraint that defines an upper bound for the price a customer is willing to pay. Bundle requests can be specified identical to bundle offers.

Agreements: Once a joint understanding is determined, i.e. the requirements specified in an offer as well as in a request are fulfilled by each other, an agreement between provider and customer is reached. The agreed functionality together with the determined properties (e.g. price, quality level) have to be fixed in a contract and communicated to the market participants. In this contract properties of a deal have to be fixed exactly, i.e. constraints should restrict the allowed values of an attribute exactly.

In the following requests and offers together are referred to as *Orders*; a set of orders as *Order Book*. All three primitives mentioned above contain capabilities of a service as well as constraints regarding the usage of the service. These two components are described in section 4.3 and 4.4.

4.3 Service Capability Description

In literature there are two distinct ways to formally represent service capabilities. Basically capabilities can be modelled explicitly and implicitly (cf. [15]). Explicit represen-

[5]Note that *Constraints* can still be defined for individual services in the bundle by using the *Object* concept in the *Constraint Description* to refer to the according capability.

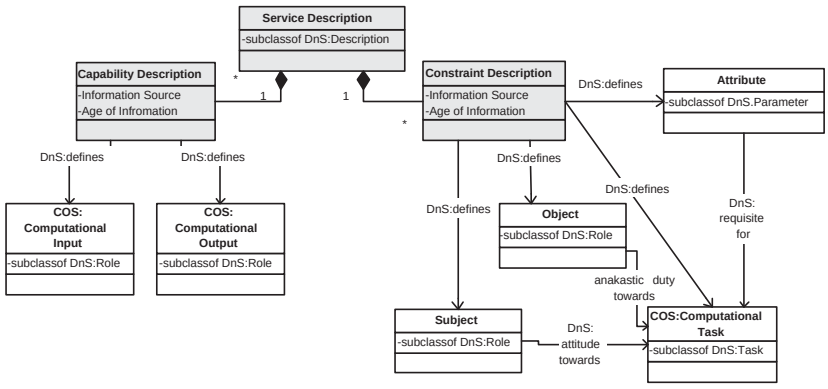

Figure 3: Sketch of the service description ontology. Concepts and associations derived from DOLCE, Descriptions&Situations(DnS), and the Core Ontology of Services(COS) are labelled with the according namespace.

tation of capabilities captures the semantics of the tasks or processes that are performed by a service, whereas implicit modelling is done by describing the transformation that is preformed by a service.

For our work we reuse a revised version of the Core Ontology of Services [12, 14, 16] which is basically suited to model both approaches mentioned above. For simplification, we use a pure implicit modelling approach, where only inputs and outputs are considered for capability matching in the marketplace.

As shown in figure 3, a *Capability Description* consists of *Computational Inputs* as well as *Outputs*. All these descriptive entities are *Roles* that are played by *Computational Objects*, e.g. a concrete message in the system. In order to allow bidding on service bundles, service descriptions may include more than one capability description.

4.4 Constraint Description

Service providers and service requesters may want to express requirements that have to be fulfilled by potential business partners. Such requirements could be constraints regarding the desired/offerd service or constraints regarding properties of providers and customers themselves, e.g. the age of a customer must be at least 18 years. In the context of Web Services these constraints are often referred to as policies. There already exists several declarative policy languages either formalized by XML, e.g. WS-Policy or XACML, or by ontologies, e.g. KAoS [17] and REI [18]. As discussed in [19], the foundational ontology DOLCE could provide the right means for integrating these existing efforts.

[19] introduces a framework for expressing policies based on DOLCE and D&S. As sketched in figure 3, a *Constraint-Description* consists of the concepts *Subject*, *Computational Task*, *Object*, and *Attribute*. The first three concepts allow to define the application area of the constraint, while subclasses of the concept *Attribute* allow to define the property of a service that is constrained. *Attribute* is *valued-by* a DOLCE

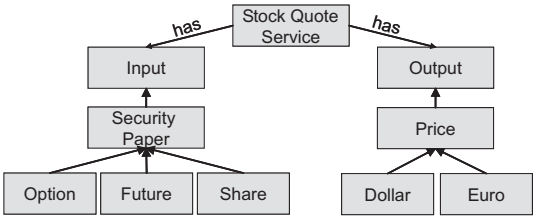

Figure 4: Stock Service

Figure 5: Offer A

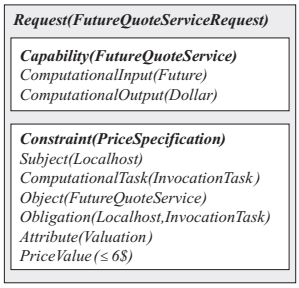

Figure 6: Request B

Region that defines the overall range of the attribute values. Values not contained in this *Region* are not allowed.

4.5 Example

As an example of the ontology based communication language, consider a Web Service returning quote requests. The capability by means of input and output descriptions of the service are shown in figure 4.

A user A is supplying this service capable of returning quotes in Dollar for any given security (i.e. options, shares, and futures). Furthermore, the user has a reservation price of 5$ for this service. Another user B is demanding a stock quote service capable of providing quotes in dollar for futures. User B valuates this service with 6$. Figure 5 and 6 show the formalization of the offer and request by means of the proposed communication language.

5 Components

In this section the main components of the marketplace already introduced in section 3 are described in detail.

5.1 Trading Component Registry

The Component Registry contains a description of each trading component. These descriptions are determined by the capabilities of the services traded in the corresponding

trading components. In order to allow forwarding of incoming orders to the right component, it has to be described by the most general inputs and outputs. For instance, a component trading the orders from example 4.5 is described by the input *Security Paper* and the output *Dollar*. Note that a component description may comprise several capability descriptions.

5.2 Market Administrator

As mentioned above, the overall marketplace is divided into several trading components. Each of them determines the allocation for a specific class or classes of services. The core functionality of the Market Administrator is to manage the trading components and to forward incoming orders to the appropriate component. In this section, we introduce the matching algorithm for comparing orders and component descriptions. Furthermore, we present management operations that are preformed by the Market Administrator to keep the set of trading components consistent.

5.2.1 Semantic order forwarding.

Once an order is received, the Market Administrator matches the capabilities of a component description derived from the registry with the capabilities contained in the order by means of a DL-reasoner. In line with [9], matching of capabilities is done by matching all inputs (in_o) of the order with all inputs (in_m) of the component description and, accordingly, all outputs of the order (out_o) with all outputs of the offer (out_m). We define two capabilities as *related* if there is a subsumption between all inputs of the component description and at least one input of the order, i.e. in_o subsumes in_m or vice versa. Analogously, for all outputs of the order there has to be a subsumption to at least one output of the component description. This can be formalized by the following (simplified) rule.

$$\begin{aligned} related(x,y) \leftarrow & capability(x) \wedge capability(y) \\ & \wedge \forall i.((in_o(i) \rightarrow in_m(i)) \vee (in_m(i) \rightarrow in_o(i))) \wedge \\ & \forall o.((out_o(o) \rightarrow out_m(o)) \vee (out_m(o) \rightarrow out_o(o))) \end{aligned}$$

The rule has to be checked for all descriptions in the registry and all included capabilities. A trading component is already *related* if on of its capabilities is *related*. In case of a bundle request or offer (i.e. more than one capability), one order might be *related* to several components. In this case the components have to be merged. In case there is no relation between the incoming order and any component description, a new market has to be created together with a new description in the repository.

5.2.2 Operations for trading component management.

As already mentioned, in order to keep the set of trading components consistent, the Market Administrator has to apply operations that allow for managing the components and their descriptions. The set of components are considered to be consistent if the allocation outcome calculated in distributed components is identical to the outcome

that would be calculated in one central mechanism. Consistency check has to be done every time an order is submitted or an agreement is reached. The following operations are needed in this context:

Create a new component: A new trading component will be created if no appropriate (related) component can be identified for an incoming order. This is done by creating an instance of a Trading Component (i.e. Preprocessor and Auction) and by adding a new component description to the registry.

Remove a component: A trading component has be removed together with the according component description if the order book is empty. Thus, the following rule has to be checked each time an agreement is reached in the trading component:

$$empty(x) \leftarrow OrderBook(x) \wedge \neg\exists y.Order(y) \wedge component(y, x)$$

Split a component: For doing so, the order book of the trading component has to be split up into several individual order books. Each of them is executed by an independent trading component. To find out whether an order book has to be divided, the following rule can be applied:

$$\begin{aligned} split(x, y) \leftarrow & OrderBook(x) \wedge OrderBook(y) \wedge Order(w) \wedge component(w, x) \\ & \wedge Order(v) \wedge component(v, y) \wedge \neg\exists z.Capability(z) \\ & \wedge component(z, w) \wedge component(z, v) \end{aligned}$$

Because the order books are divided only in case they contain two fully distinct sets of service types, the split operation does not lead to any market defects (e.g. missed matches, decreasing liquidity). Imagine a trading component contains stock quote as well as currency translation services. Obviously, the order book can be split into one book containing only stock quote and one containing only translation services without loosing potential matches. Trading components should be divided as soon as possible to reduce the complexity of the allocation mechanism.

Merge components: Several trading components are integrated by removing all components except for one, which contains the merged order book. Merging becomes necessary when a new order arrives that is related to capabilities traded in two different components. This will be the case if more than one matches (direct or indirect) between component description and order:

$$\begin{aligned} merge(x, y) \leftarrow & OrderBook(x) \wedge OrderBook(y) \wedge Order(w) \\ & \wedge component(x, w) \wedge Order(v) \wedge component(y, v) \\ & \wedge \exists z.Capability(z) \wedge component(w, z) \wedge component(v, z) \end{aligned}$$

Suppose there are two markets, one dealing with stock quote and one with currency translation services. If a request for a financial service arrives, these two markets have to be merged.

Update a component description: Each time a order is forwarded or an agreement reached the component description has to be updated. This is done by determining the most general input and output for all capabilities in the description.

5.3 Trading Component

Each instance of a Trading Component consists of two building blocks: A Preprocessor and an Auction component. These components are described in the following.

5.3.1 Preprocessor

Due to the fact that two services could match semantically, although their syntactic descriptions are not fully identical, a **semantic preprocessing** step is needed to assure that all possible matchings are considered when calculating the allocation. Afterwards, the order will be transformed to the auction's bidding language by a **syntactic preprocessing** step.

An offer will *semantically match* a request if all inputs and outputs match. Inputs match in case all inputs of the offer subsume at least one input of the request. Outputs match in case all outputs of the request subsume at least one output of the offer. This is formalized in the following simplified axiom, where (in_r, out_r) represent the inputs and outputs of the request c_r and $(in_m, outm)$ those of the offer c_o.

$$semantic - match(c_r, c_o) \leftarrow \forall x.(in_r(x) \rightarrow in_o(x)) \wedge \forall y.(out_r(y) \rightarrow out_o(y))$$

Note that the objective of this preprocessing is not to match bids, but to transform the order book in a way that the results of the syntactic matching is identical to the semantic matching defined above. Syntactic matching can be defined as follows:

$$syntactic - match(c_r, c_o) \leftarrow \forall x.(in_r(x) \leftrightarrow in_o(x)) \wedge \forall y.(out_r(y) \leftrightarrow out_o(y))$$

Based on this definitions, the orders mentioned in example 4.5 match only on a semantic level due to the different degree of abstraction of the input concepts. In order to get the same matches in a syntactic algorithm as in a semantic approach, the ontology is used to generate alternative bids. These bids can then be concatenated by a XOR operator[6], ensuring that at most one bid will be executed. Therefore, the following rules can be used:

- If a concept X that represents the input of a request (e.g. Future) is subsumed by another concept Y in the ontology (e.g. Security Paper), a new XOR-associated request has to be introduced with input Y. This has to be done for each concept that subsumes X and is subsumed by the input of the component description. This would mean for request B of the example 4.5 that the alternative request $(SecurityPaper, Dollar)$ has to be introduced.
- If a concept X that represents the output of an offer (e.g. Dollar) is subsumed by a concept Y in the ontology (e.g. Price), a new XOR-associated offer has to be introduced containing output Y. This has to be done for each concept that subsumes X and is subsumed by the input of the component description. This means, for Offer A of the example, a new offer $(SecurityPaper, Price)$ has to be added.

[6] A XOR B means either A, B, or $\emptyset$, but not AB.

- In order to avoid unnecessary large concatenations of bids, the Preprocessor has to check each time an agreement is reached, if the trading component description is is changed to a more specific one. If this is the case, the introduced bids that are more general than the component descriptions have to be removed.

This results in two alternative requests $(SecurityPaper, Dollar)$ and $(Future, Dollar)$ as well as two offers $(SecurityPaper, Price)$ and $(SecurityPaper, Dollar)$. Now, it is easy to see that there will also be a match in the syntactic approach. The auction mechanism has to be able to guarantee that only one alternative request and offer is accepted (XOR-bids).

In case of semantic matching, it can be shown that after applying these rules the possible matches are identical compared to the situation without applying the rules. In case of syntactic matching, applying the rules could significantly improve the matchmaking (as illustrated in the example), while it is guaranteed that at least the same matches are found compared to situation without applying the rules.

After the semantic preprocessing, the **syntactic preprocessing** is applied. Here, the semantic order book is translated into the bidding language of the auction mechanism. For the auction mechanism, a buyer order has to be transformed into a XOR concatenated set of bundle bids $B_{n,1}(S_1) \oplus \ldots \oplus B_{n,u}(S_j)$, where u is the number of bundle bids in the order. A single bundle bid $B_{n,f}(S_i)$ is defined as the tuple

$$B_{n,f}(S_i) = (v_n(S_i), (q_n(S_i, g_1, a_{g_1,1}), \ldots, q_n(S_i, g_j, a_{g_j,A_j}))).$$

In the bundle bid, S_i denotes any bundle (e.g. a stock quote service and a weather service) and $v_n(S_i)$ defines the valuation for this service. Furthermore, the attribute characteristics can be expressed by $q_n(S_i, g_i, a_{g_j,k})$ where $g_i \in S_i$ is a specific service and $a_{g_j,k} \in g_i$ is an attribute of it.

The orders of the sellers are formalized in a similar way as the buyers' orders are. An order is defined as a concatenated set of XOR bundle bids $B_{m,1}(S_1) \oplus \ldots \oplus B_{m,u}(S_j)$, where u is the number of bundle bids. A single bid $B_{m,f}(S_i)$ for a bundle S_i is defined as the buyer's bids are, except the valuation $v_n(S_i)$ is replaced by a reservation price $r_m(S_i)$.

As an example of the syntactic preprocessing the orders will be transferred into the bidding language required by the auction mechanism:

The requests $(SecurityPaper, Dollar)$ and $(Future, Dollar)$ are represented as two XOR concatenated bids $B_{m,1}(SecurityPaper) \oplus B_{m,2}(Future)$ with

$$B_{m,1}(SecurityPaper) = (6, Dollar) \text{ and } B_{m,2}(Future) = (6, Dollar).$$

The XOR operator ensure, that at least one bid will be executed. The offers will be transformed analogously to $B_{n,1}(SecurityPaper) \oplus B_{n,2}(SecurityPaper)$ with

$$B_{n,1}(SecurityPaper) = (5, Dollar) \text{ and } B_{n,2}(SecurityPaper) = (5, Dollar).$$

5.3.2 Auction Mechanisms

The auction mechanism used in the architecture is a multiattribute combinatorial exchange as proposed in [8]. The auction fulfills most of the requirements given in section

2, as the mechanism allows multiple buyers and sellers simultaneously (R2) the submission of heterogonous bundles expressing (R3,R5) either substitutabilities (realized by XOR bids) and complementarities (realized by bundle bids). Furthermore, the mechanism is capable of handling cardinal attributes (R6) as well as an immediate execution of given orders as the clearing can be done continuously[7] (R1). Besides a syntactically based bidding language, the auction subsumes a winner determination component (i.e. allocation of services from suppliers to requesters) and a pricing schema (i.e. which price have to be paid on base of the allocation).

The winner determination problem maximizes social welfare, i.e. the difference between the buyers' valuations and the sellers' reservation prices. The winner determination problem is formulated as a linear Mixed Integer Programm (MIP) and thus can be solved by optimization solvers such as CPLEX[8]. The winner determination is, however, an generalization of the combinatorial allocation problem (CAP) and thus $\mathcal{NP}$ complete [20].

The winner determination alone is insufficient to ensure an efficient outcome of the mechanism. Thus, a pricing mechanism is required which reveals the true internal information of the participants about the demand and supply situation (i.e. preferences). In the proposed auction, an approximated Vickrey schema is used which results in an approximated allocative efficiency [20].

6 Performance Simulation

As mentioned above, the proposed architecture is capable of splitting markets, if two or more disjunctive order sets exist. This may lead to high performance gains especially if the auction mechanism is computational demanding. In the following, this performance gain is measured by means of a stochastic based simulation.

An order stream is generated which consists of two disjunctive order subsets. Each buyer and seller submits five bundle bids as an order. The bundles are generated using the Decay distribution, which is shown to lead to hard instances of general combinatorial allocation problems [21].

In the first scenario, the order stream is computed using a single instance of the mechanism. In the second scenario, the order stream is split into two disjunctive subsets and computed using two independent instances of the mechanism. Figure 7 shows the computation time of CPLEX for calculating the allocation in both scenarios. In scenario 2, the computation time of the independently computed subsets are added to make the overall computation time comparable with the first scenario.

Due to the exponential runtime of the auction mechanism, it is obvious that an intelligent order book splitting leads to performance gains. For instance, the winner determination for 900 bundle bids takes more than 140 seconds in scenario 1. Splitting the subsets and computing the order books separately (each with 450 bundle bids) takes only 40 seconds.

[7] Continuously clearing means a clearing each time a new order enters the market.

[8] CPLEX is a commercial solver for optimization problems (`http://www.cplex.com/`).

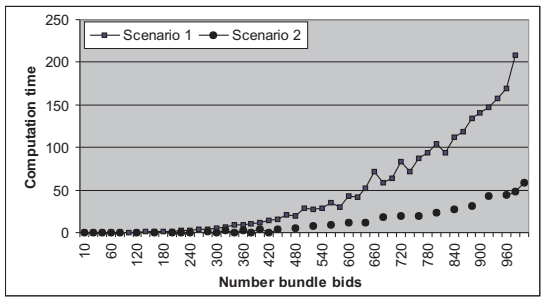

Figure 7: Performance evaluation

However, the results cannot be generalized as the computation time for the intelligent order splitting is neglected at the moment. Nevertheless, the simulation shows that the use of background knowledge may reduce the overall complexity.

7 Related Work

For maintaining a loose coupling between service requester and provider, dynamic service discovery plays a crucial role. Thus, several algorithms and frameworks have been proposed to tackle this problem. Some of them are based on syntactic service descriptions, like description repository UDDI or the discovery protocol WS-Discovery[9]. Others, like [9, 22, 23], suggest to adopt semantic service descriptions for matchmaking. However, while providing semantic matching capabilities, these algorithms use centralized matching components without the employment of prices. Thus, these algorithms require full information about the demand and supply situation in order to be effective. However, this information requirement is not even closely met [20, 7].

Market-based approaches incorporate incentives for truthful information revelation by implementing prices. [5] motivate exemplarily the use of price systems for allocating Web Services. However, complementarities, attribute characteristics, and semantics based order specifications are not considered and, thus, the mechanism does not fulfill the defined requirements in section 2. Furthermore, several architectures have been introduced for an economic allocation of resources in similar domains such as the Computational Grid, e.g. [24]. However, these mechanisms are not directly applicable for trading Web Services as they do not consider the dependencies between services (i.e. by the use of bundles) and rely on attribute-value based bidding languages.

There are several bilateral mechanism schemas for negotiating services. For instance, [6] implement a semantic-enabled negotiation system. However, the systems does not allow the simultaneous trading of multiple buyers and sellers and thus do not fulfill the requirements specified in section 2.

Besides, ontologies cannot only be used to improve matching within a market as done in this work, but they can also be used for modelling negotiation protocols and

[9] See `http://www.uddi.org/`, http://msdn.microsoft.com/ws/2004/10/ws-discovery/ for details.

strategies [25]. This could allow agents to adapt to different market mechanisms. However, this work is complementary to ours and becomes relevant when considering different mechanisms in the market.

8 Conclusion

This paper outlined the design of an ontology-driven market for trading Web Services. Based upon a requirement analysis for a Web Service market, a marketplace was designed which is up to these marks. The marketplace uses an ontology based communication language that is capable of representing semantically described request, offers, and agreements. These ontology-driven messages were transformed into syntactically represented orders so that an existing auction mechanism could be used while still allocating on a semantic level. Furthermore, semantic information was used to split the whole market into several independent sub-markets. The concept was shown to be more efficient than an existing mechanism, i.e. the use of background knowledge has reduced the overall complexity.

For the future, additional auction mechanisms (e.g. English auctions) will be integrated for making the overall platform generally applicable. Furthermore, the existing platform will be compared and benchmarked with semantics-based matchmakers as well as classical auction marketplaces in more detail.

References

[1] Lawler, J., Anderso, D., Howell-Barber, H., Hill, J., Javed, N., Li, Z.: A study of web services strategy in the financial services industry. In: The Proc. of ISECON 2004. (2004)

[2] Veit, D.: Matchmaking in Electronic Markets. Springer Verlag (2003)

[3] Harth, A., Decker, S., He, Y., Tangmunarunkit, H., Kesselman, C.: A semantic matchmaker service on the grid. In: WWW (Alternate Track Papers & Posters). (2004)

[4] Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proc. of 12th Int. WWW Conf., New York, NY, USA, ACM Press (2003)

[5] Li, Z., Zhao, H., Ramanathan, S.: Pricing web services for optimizing resource allocation an implementation scheme. In: Proc. of the Web2003, Seattle, WA. (2003)

[6] Vasiliu, L., Zaremba, M., Moran, M., Bussler, C.: Web-service semantic enabled implementation of machine vs. machine business negotiation. Techreport, DERI Innsbruch (2004)

[7] Neumann, D.: Market Engineering - A Structured Design Process for Electronic Markets. PhD thesis, Economics and Business Engineering, University of Karlsruhe (TH) (2004)

[8] Schnizler, B., Neumann, D., Weinhardt, C.: Resource allocation in computational grids - a market engneering approach. In: Proc. of the WeB 2004, Washington. (2004)

[9] Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: Proc. of the 1st Int. Semantic Web Conf., Springer (2002)

[10] Staab, S., Studer, R.: Handbook on Ontologies. Springer Verlag, Heidelberg (2004)

[11] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: The WonderWeb library of foundational ontologies. WonderWeb Deliverable D17 (2002)

[12] Mika, P., Oberle, D., Gangemi, A., Sabou, M.: Foundations for service ontologies: Aligning owl-s to dolce. In: Proc. of the 12th Int. Conf. on WWW, ACM (2004)

[13] Gangemi, A., Mika, P.: Understanding the semantic web through descriptions and situations. In: Confederated Int. Conf. DOA, CoopIS and ODBASE. LNCS, Springer (2003)

[14] Gangemi, A., Mika, P., Sabou, M., Oberle, D.: An ontology of services and serivce descriptions. Technical report, Laboratory for Apllied Ontology, Rome, Italy (2003)

[15] Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics **1** (2003)

[16] Oberle, D., Lamparter, S., Eberhart, A., Staab, S.: Semantic management of web services. Technical report, University of Karlsruhe (2005)

[17] Uszok, A., Bradshaw, J.M., Jeffers, R., Tate, A., Dalton, J.: Applying KAoS services to ensure policy compliance for semantic web services workflow composition and enactment. In: Int. Semantic Web Conf. (ISWC'04). (2004)

[18] Kagal, L.: A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. PhD thesis, University of Maryland, Baltimore MD 21250 (2004)

[19] Lamparter, S., Eberhart, A., Oberle, D.: Approximating service utility from policies and value function patterns. In: 6th IEEE Int. Workshop on Policies for Distributed Systems and Networks, IEEE Computer Society (2005)

[20] Parkes, D.C., Kalagnanam, J., Eso, M.: Achieving budget-balance with vickrey-based payment schemes in exchanges. In: Proc. of the 12th Int. Joint Conf. on AI. (2001)

[21] Sandholm, T., Suri, S., Gilpin, A., Levine, D.: Winner determination in combinatorial auction generalizations. In: Proc. of 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems, ACM Press (2002)

[22] Herzog, R., Lausen, H., Roman, D., Stollberg, M., Zugmann, P.: WSMO Registry. Technical report, DERI (2004)

[23] Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: Meteors wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. Journal of Information Technology and Management (2004)

[24] Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic models for resource management and scheduling in grid computing. J. of Concurrency and Computation: Practice and Experience **14** (2002)

[25] Tamma, V., Phelps, S., Dickinson, I., Wooldridge, M.: Ontologies for supporting negotiation in e-commerce. Engineering Applications of Artificial Intelligence (2005)