

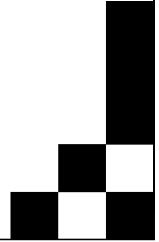
A b s t r a c t

Electronic markets and especially online auctions gain more and more importance and a plethora of market mechanisms is emerging on the Internet. The great variety of trading rules hinders agents easily switching between different marketplaces and, therefore, fragments the overall market – supply and demand might fail to meet due to agents' inability to interact with the same market mechanism. The problem can be overcome by a *descriptive auction language* (DAL) allowing for the machine-readable specification of arbitrary auction mechanisms. The implications are twofold: on the one hand a market engineer can coherently describe a mechanism by means of the language and automatically deploy it via an auction runtime environment and, on the other hand, a (software) agent can automatically deduce valid and reasonable actions from the description of a previously unknown auction mechanism.

Keywords: auctions, trading software agents, Internet economics, market engineering, e-commerce, e-business

A Descriptive Auction Language

DANIEL ROLLI, STEFAN LUCKNER, HENNER GIMPEL AND CHRISTOF WEINHARDT



INTRODUCTION

Economic transactions are ubiquitous in everyday life: people constantly face decisions to purchase goods and services or to sell them to others. Thereby, posted price offers are a widespread mechanism, but even this simple mechanism is oftentimes accompanied by more or less structured negotiations. Furthermore, there are diverse auction types, exchanges and the like. The plethora of market mechanisms is infinite.

Humans can easily switch between different market mechanisms – they are used to ask for the rules of the encounter, to interpret natural language descriptions and to successively adapt their behaviour by experience. The better one knows a mechanism, the easier it is to find valid and reasonable behaviour and to conduct transactions according to one's preferences. Even if a mechanism is entirely new, humans are able to reason on the rules and to conduct transactions via the new mechanism. For software agents it is not that easy.

The code of a software agent unambiguously confines what the agent is capable of. Many agents are specifically tailored to one market mechanism: they can bid on eBay, they can surf through a pre-defined set of online-shops, they can find flights from major airlines, etc. But they cannot adapt to unknown

A u t h o r s

Daniel Rolli

(daniel.rolli@iw.uni-karlsruhe.de) is a PhD student and works in the interdisciplinary project SESAM. His research interests include market description methods, ephemeral markets, secure decentralized market mechanisms and adaptive trading agents.

Stefan Luckner

(stefan.luckner@iw.uni-karlsruhe.de) is a PhD student in the STOCER project and member of the Graduate School 'Information Management and Market Engineering'. His research focuses on marketplace architectures, service-oriented computing and forecasting markets.

Henner Gimpel

(henner.gimpel@iw.uni-karlsruhe.de) is a

PhD student working on negotiation analysis, auction theory, service-oriented computing, behavioural economics and experimental economics.

Christof Weinhardt

(christof.weinhardt@iw.uni-karlsruhe.de) is full professor for Information Management and Systems and member of the Research Center for Information Technologies (FZI) in Karlsruhe, Germany. His research interests include electronic markets, auction and negotiation theory, multi-agent systems, market engineering and incentive engineering.

All four authors are affiliated with the Institute of Information Systems and Management in the School of Economics and Business Engineering at the University of Karlsruhe, Germany.

market mechanisms they were not implemented for. The diversity of mechanisms employed in e-business is huge and new ones are frequently envisioned. Therefore, software agents are likely to come across unknown mechanisms and to fail in closing a desirable deal because the mechanism cannot be handled. Today, the task of coping with different and potentially unknown market mechanisms is left to humans as there is no general coherent machine-readable description format.

How is a software agent supposed to figure out what to do in an e-business marketplace? What actions are allowed, what behaviour is reasonable, what is desired by the principle? What are the common structures inherent even in unfamiliar market mechanisms? These are the questions addressed in this paper. We aim to reveal the primal constitution of auctions and to derive the coherent and unambiguous specification of auction mechanisms. This constitutes a significant step towards making even new mechanisms interpretable for software agents, which is vital for improved convenience in Internet transactions. To this end, we outline the basic reasoning of a software agent deducing valid and reasonable actions. The design and implementation of agents that are capable of fully interpreting the language presented here is, however, still future work.

Furthermore, a market engineer has to make hundreds and thousands of design choices when building a new market platform from scratch (Weinhardt *et al.* 2003). The engineering task can be eased by a runtime environment for automatically deploying markets specified in a machine-readable way. The *descriptive auction language* (DAL) we present builds on SQL and XML to smoothly describe and implement new auction mechanisms on the one hand and automatically understand the rules of the encounter and deriving reasonable actions on the other hand. The expressiveness of DAL thereby exceeds known parameterization approaches (e.g., Lochner and Wellman 2004; Ströbel and Weinhardt 2003) and supplements the description with a data model.

The following section presents related work. We then introduce the architecture of a market server for running an auction based on its DAL description. Subsequently, the descriptive auction language itself is presented and the basic structure of software agents utilising it is sketched. The final section concludes.

RELATED WORK

Parameterization approaches are the closest related work in modelling auctions, although DAL is not a parameterization approach itself. Many of these approaches are related to an auction server running the auction that is specified by the respective parameters. Three of them are presented in the following. The Michigan Internet AuctionBot is a well-known example of a configurable

marketplace employing an auction description format (Wurman *et al.* 2002). The employed parameterization of auction rules is organized according to three basic activities of auction mechanisms: handle bid requests, compute exchanges and generate intermediate information. This goes well beyond early taxonomies by, for example, Engelbrecht-Wiggans (1980) and Friedman (1993). The AuctionBot specifies auction mechanisms by assigning values to parameters relating to these three basic activities (Wurman *et al.* 2001). The restrictiveness of the AuctionBot's parameterization – as well as any other parameterization – was one motivation for the development of the AB3D server. To allow for the introduction of issues for innovative auction designs Lochner and Wellman (2004) augment the parameterization of Wurman *et al.* (2001) by a rule-based scripting language. AB3D thereby inherits the three basic activities from AuctionBot. Lochner and Wellman (2004) describe that parameter-based approaches are especially ill-suited to capture the temporal pattern of an auction and, thus, introduce rule patterns to model the control structure of an auction. This is related to constructs that can be modelled in DAL, for example the condition-action term in Code Listing 8.

In our DAL approach, three components – validation, agreement generation and view – take up the respective activities of the AuctionBot and AB3D. In contrast to the latter, DAL components are constructed as self-contained components with a defined interface to the shared foundation of the data and come with a coherent description language for each. The present DAL approach goes further with respect to the flexibility of descriptions, as it does not rely on any parameterization of the auction design space in the first place where AB3D retains a parametric approach for many basic features of an auction. A second difference is that DAL explicitly provides a data model that is always necessary for building new auction rules. AB3D certainly also draws on a similar model, but it is not presented by the authors.

The limitation of the (partially) parameter-based approach of AB3D exemplifies in the pricing rule. Lochner and Wellman (2004) use the parameter *pricing k* in several of their examples. K-double auctions are a prominent class of mechanisms in academic literature and some real-world markets, but nevertheless they are just one class of mechanisms. Many stock exchanges, for example, utilize price-continuity rules. A double auction is easily imaginable where the price is set equal to the last transaction price if this last price falls into the intersection of bid and ask price. A price-continuity pricing rule like this cannot be modelled with the *pricing k* parameter; once more a next parameter must be added for a variation of the same issue. If such an example is given, a parameterization approach can always be extended ex-post. However, the crucial point is that it will never be possible to foresee all relevant parameters.

The meet2trade system is a generic market platform for parametric design of auction-based electronic markets (Weinhardt *et al.* 2005). A market modelling language is utilized for two purposes: on the one hand it provides a taxonomy of parameters for configuring various auction families and on the other hand the modelling language is used for combining simple market mechanisms to so called meta-markets. Again, the parametric foundation of specifying auctions facilitates the configuration of known auction types in a relatively simple way. But still, all added or renamed parameters and their effect have to be understood offline, and by definition this approach lacks the ability to represent entirely new mechanisms.

Concerning agents capable of accessing different marketplaces, Anthony and Jennings (2003) take a step beyond agents restricted to one specific market mechanism. They propose an autonomous agent that can participate in multiple heterogeneous auctions. In their setting, they have several concurrent instances of three types of auctions – English, Dutch and Vickrey. An agent decides which auction to participate in and what bid to submit. As the authors mention, this approach is strictly limited to predefined types of auction mechanisms and strategies, which are specified at design time. In contrast to DAL, a structural description of the mechanisms is not provided to the agent and new auction mechanisms cannot be understood at runtime.

MARKET ARCHITECTURE

We consider two perspectives on an auction: the server running the auction and handling bids as well as the individual participants with their preferences and strategies. Providing a new auction mechanism should not result in a software developer having to write the program code by hand as this can be a long and tedious process. In fact, executable implementations of arbitrary auction mechanisms can directly be generated from their description in DAL. Figure 1 gives an overview of our approach.

Figure 1 sketches three roles for users of the DAL market server: a market engineer specifies an auction with DAL, a market initiator starts one particular instance of a previously specified auction, and a market participant bids in such an auction instance. Of course, one single user might take several roles.

When receiving a DAL document, the server can generate the actual implementation of the auction mechanism according to its description and store the generated code in a repository. From then on, new auctions using this mechanism can be deployed by a market initiator.

Market participants – either human or software agents – may search for auction instances by means of the market registry. Before trading on the market, participants may request the description of a suitable auction

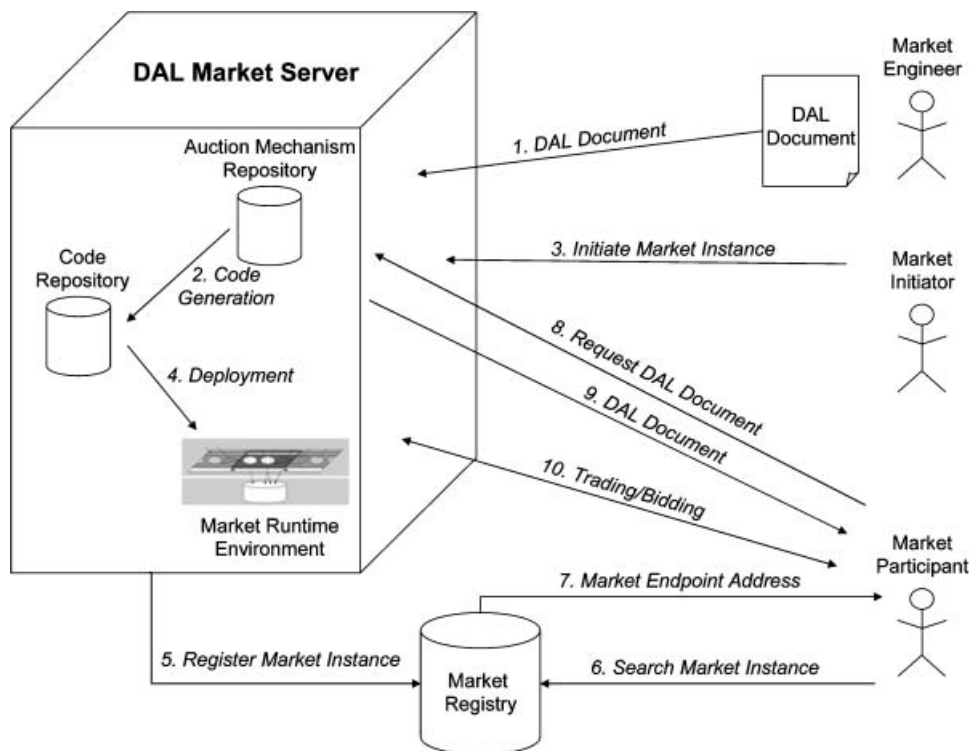


Figure 1. Market architecture

mechanism from the auction mechanism repository. Previous work on trading software agents tailors the agents' behaviour to each particular mechanism in advance (Anthony and Jennings 2003; Cheng *et al.* 2005). Such agents themselves cannot examine the functioning of an auction mechanism at runtime. On the contrary, DAL facilitates software agents assessing previously unknown auction mechanisms at runtime.

Recapitulating, the descriptive auction language makes up the core of our market architecture. Its end-to-end use comprises designing auction mechanisms, generating code, deploying market instances and finally deriving market participants' behaviour. This does not imply that building an auction mechanism is a straightforward process – it rather is a complex engineering task (Weinhardt *et al.* 2003). However, once the economic properties and rules are defined, DAL eases implementing and deploying a mechanism.

DESCRIPTIVE AUCTION LANGUAGE

In the following, the data model and the auction reference model underlying DAL are outlined and then several auction specifications according to DAL are exemplified.

Market data model

Markets are information processing systems. The market participants' offers, the agreements reached and timing information need to be stored in a market database which can be accessed from within a DAL document. We refer to the kind of data that is collected while running a market as *market instance data*. Its structure draws on a minimal market model which builds a market vocabulary based on common structures of markets and market mechanisms (Rolli *et al.* 2004). Such a vocabulary is needed in order to facilitate the communication of market participants. For DAL, it comprises the concepts *participant*, *intention*, *product*, *attribute*, *agreement* and *offer*. A *participant* represents an agent that takes part in a market. Market participants submit intentions to a market. An *intention* thereby represents the smallest closed entity of purpose and is defined by two groups of products. Incoming products are the ones the participant wants to receive while outgoing products are the ones he is willing to give away in exchange. *Products* may be any good or service. *Attributes* are used to describe properties of products and market participants. Each *offer* contains one or several intentions and serves as a container of control data for communicating these intentions. Offers facilitate exclusive or (XOR) relations between intentions, i.e. only one of the intentions in an offer can lead to an agreement. An *agreement* requires matching associated intentions.

Figure 2 depicts the data model for the relevant market data of an auction instance following the Structured Entity Relationship Model (SERM, Sinz 1988). Connections between object types correspond to relationships in the standard Entity Relationship Model (ERM).

Participants submit offers enclosing XOR combinations of intentions to a market. Thereby, an intention is related to at least one incoming and one outgoing product. Relating several products to one intention allows for defining AND combinations of these products. The maximum and minimum amount of every product instance is specified by two separate attributes. Note that money is also considered a product in this modelling. In order to facilitate partial execution where desired, the primary key of an intention is a combination of an intention number and a timestamp. After partial execution, the remaining part of an intention is stored as an intention with the same intention number but a new timestamp. Overwriting an existing intention is not possible as all original information shall remain available for later access.

All relevant product data can be specified by appending attributes. Thereby, attributes can either be text, numbers or a range of numbers that is described by an interval. Attributes with identical names of the same product define a set of XOR choices for the respective value. The same holds for attributes describing market participants. Finally, agreements that comprise pairs of matching intentions can be stored.

The data model is universal in the sense that it is not restricted to a certain mechanism. In fact, the description of any auction mechanism in DAL is based on this data model and comprises a number of SQL operations accessing the resulting market instance data as well as a number of XML constructs concatenating the SQL statements.

Auction reference model

Based on the data model, the structure of DAL follows the notions of an auction reference model (ARM, Rolli and Eberhart 2005). The ARM provides a descriptive foundation for structuring auction mechanisms. Therefore, it decomposes them into components, which prepares the ground for systematic analysis, immediate derivation of implementation and in particular building implementations by recombining predefined components. The latter recombination 'in the large' does not require any programming skills from a market engineer.

Moreover, a full description of components, their interrelations and their embodiments delivers a cohesive specification of an auction mechanism. This has been verbally exemplified by Rolli and Eberhart (2005) and is further elaborated here by means of DAL in order to deliver a comprehensive specification for each component.

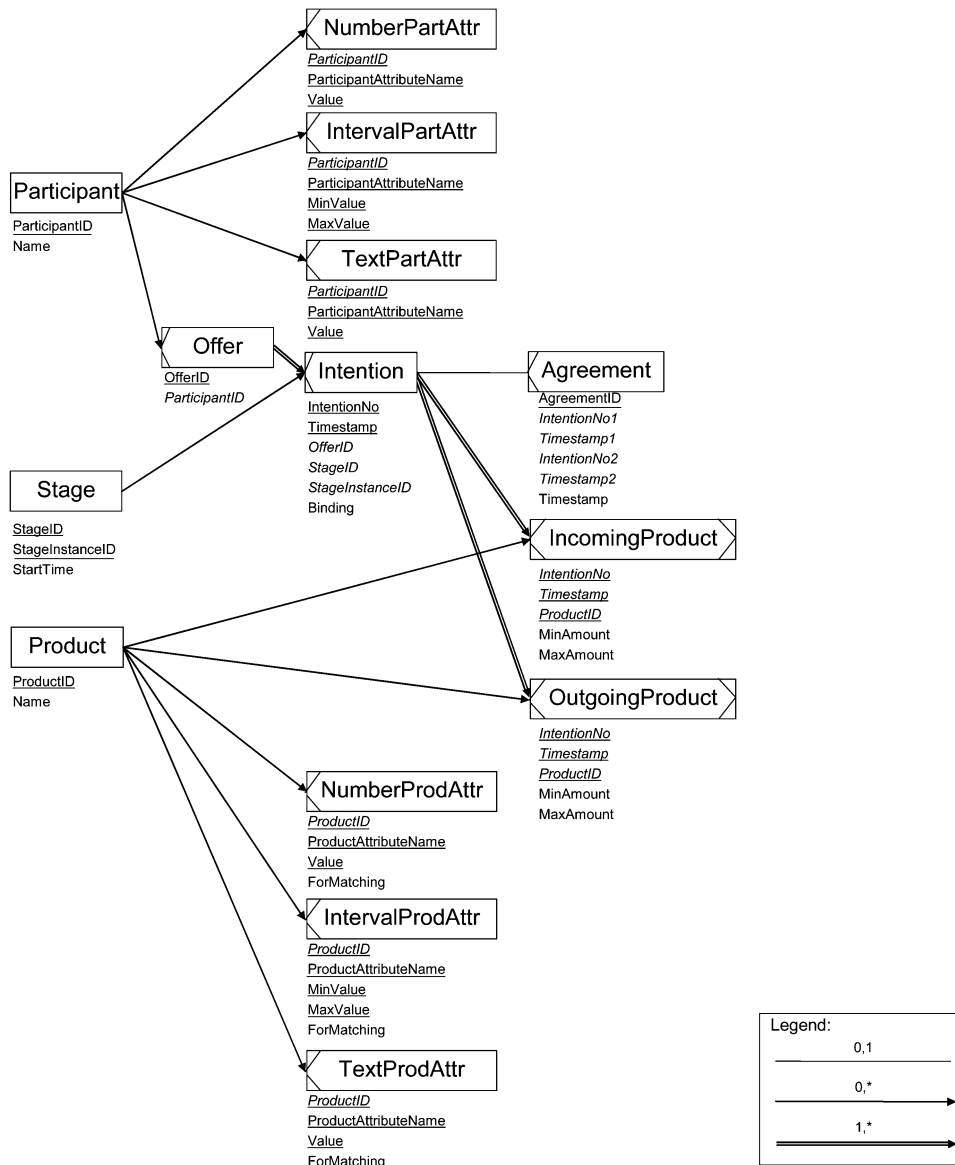


Figure 2. Market data model

The *auction data* layer at the bottom of Figure 3 forms the fundament of the ARM and follows the data model outlined before; the *auction participants* layer comprises the agents participating in the auction – humans as well as software agents.

The central position of the *auction mechanism* layer reflects its vital function. While participants – with their preferences and resulting strategies – are part of the economic environment and the data basis is common to all auction mechanisms, the mechanism itself is the sole connection of participants to the data. The mechanism controls the visibility of data to participants as well as the participants’ ability to insert data. It is this auction mechanism that a market engineer defines via DAL to coherently capture the character of each particular auction.

The auction mechanism is illustrated in the mid-layer of Figure 3 and, like any auction mechanism, assembled

from several stages each of which is an ensemble of four component types: view, validation, transition and agreement generator. These components and their DAL representation are outlined in the following section. There is exactly one stage active at any time during the auction and the primary purpose of each stage is to activate its enclosed components. The order in which the stages are executed is determined by the auction flow. In Figure 3 the active stage is displayed in a dark grey rectangle. The adjacent rectangles indicate the stages chronologically before and after the active one.

The components of auction mechanisms in detail

For the upcoming description of the auction flow and the four components: view; validation; transition; and

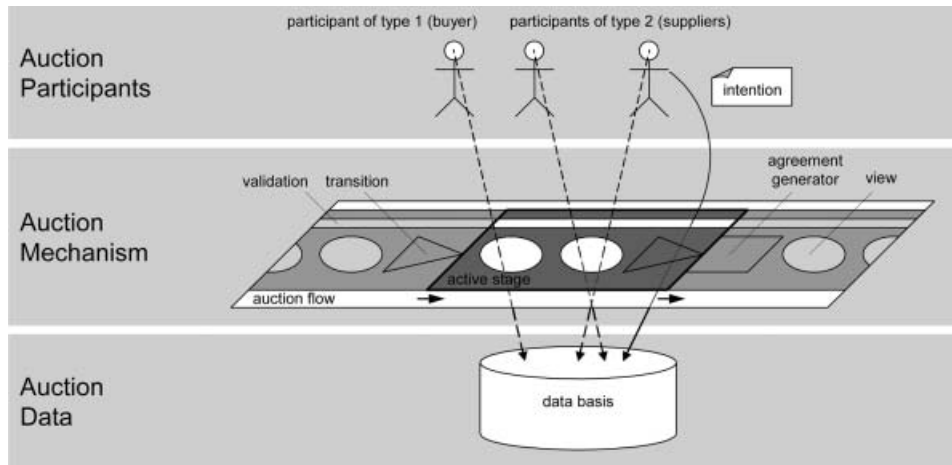


Figure 3. Auction reference model overview *Source:* Rolli and Eberhart (2005)

agreement generator we make use of an example auction to illustrate the vital aspects. In this section, we draw on the first-price sealed-bid (FPSB) auction before we subsequently demonstrate how such an auction mechanism can be systematically transformed into other types like the English, Dutch or Anglo-Dutch auction; see McAfee and McMillan (1987) and Klemperer (1998) for a verbal definition of the aforementioned auctions. With more variety in details and increasing complexity of auctions, the strengths of DAL become more prominent. The systematic derivation of auctions from FPSB to Anglo-Dutch can certainly not give an extensive insight on this but rather an initial notion. It intends to deliver the basic idea behind systematic derivation and construction of auctions with DAL.

The FPSB auction is characterized by a number of bidders competing for one good at stake. The winner who offered the best price pays the amount he has bid. No information about the competing bids is revealed to the bidders during the bidding phase. A FPSB auction can be divided into three stages. Stage 1 accepts the seller's intention to give away his product in return for money. Stage 2 subsequently accepts intentions by bidders who are interested in the product. Stage 3 generates an agreement by identifying the buyer with the highest bid as the winner who pays his price. All three stages are executed in sequence.

As any type of component can access the current state of the market instance data, we make use of a data query language. For its popularity, maturity and intuitive comprehensibility, we draw on the Structured Query Language (SQL) to model this aspect. However, as we do not require highly sophisticated queries, other query languages might as well be applied. The SQL-based data access is embedded in an XML-based structure.

Auction flow. All stages with their components are coordinated by an *auction flow*. The flow determines the

order of stages and thereby allows for loops. There is no delay between stages, i.e., a stage is either immediately followed by its successor or is the last one in an auction mechanism. Any stage is fully characterized by its embodiment, i.e. the components it comprises.

Code Listing 1 exemplifies the XML representation of an auction flow: an auction contains one or more stages and each stage contains elements of six different types. Data in the data basis can be accessed by means of `sqlQuery`s, data of a newly arriving intention via `validationQuery`s. There can be an arbitrary number of these query tags. Within each stage, the basic SQL queries can be referenced from all its components whereas a `validationQuery` can only be referenced from a `validation`.

```
<auction>
  <stage id='1'> ... </stage>
  <stage id='2'>
    <sqlQuery name='sellerID'> ... </sqlQuery>
    <sqlQuery name='highestBid'> ... </sqlQuery>
    <validationQuery
      name='submittedIntentionIncomingProduct'>...
    </validationQuery>
    <agreementGenerator> ... </agreementGenerator>
    <view> ... </view>
    <view> ... </view>
    <validation> ... </validation>
    <transition> ... </transition>
  </stage>
  <stage id='3'> ... </stage>
</auction>
```

Code Listing 1. Auction flow example

The queries are followed by at most one `agreementGenerator`, an arbitrary number of `views` and `validations` as well as one `transition`. The stages are either executed in sequence, or while-loops can be included in the auction flow.

View. Information feedback is a pivotal factor for market participants interacting with an auction. The maximum feedback each participant can obtain is determined by the *views* of an auction. In each stage there can be any number of views greater or equal to zero. In Figure 3, views are depicted as circles. They are active from the beginning to the end of the respective stage. To provide different information to different participants, the mechanism assigns different views to participants in different roles, like, e.g., sellers or buyers.

A view takes the market instance data as input, executes select statements and is capable of doing simple operations on that data. The final result is returned to the participant, who requests information on the auction's current state. Changes in the market data basis are immediately reflected in a view. On an abstract level, a push or pull implementation of views makes no difference. In practice, however, primarily pull mechanisms have been implemented for auctions that follow DAL principles, since with infrequent requests by agents it reduces the computation and communication load of the market server. The cost of less convenience for the market participants can well be put up with in a prototypical implementation. Hence, the approach we now provide takes a pull perspective but can still be analogously applied to a push alternative.

Checking a participant's authorization to access a view can be done by a SQL SELECT statement (see Code Listing 2); the following one belongs to a view in stage 2 of the first-price sealed-bid auction example. It defines the seller from stage 1 as the participant who is allowed to access the respective view:

```
SELECT ParticipantID FROM Offer WHERE OfferID=
(SELECT OfferID FROM Intention WHERE StageID=1)
```

Code Listing 2. SQL statement for identifying the seller – sellerID

Code Listing 2 is placed in stage 2 of the auction flow as sqlQuery named sellerID. It can then be referenced from within the components. Every statement for a role must deliver a set of ParticipantIDs. The runtime environment then checks if the identification of the participant demanding the view is contained within this set. If so, the respective content of the view is returned to the participant. Code Listing 3 exemplifies a view referring to the result of the above SQL statement.

```
<view>
  <checkRole>
    <queryResult queryName='sellerID'>
      <attribute name='ParticipantID'/>
    </queryResult>
  </checkRole>
  <dataOutput>
    <queryResult queryName='highestBid'>
```

```
      <attribute name='price'/>
    </queryResult>
  <queryResult queryName='numberOfBids'>
    <attribute name='numberValidBids'/>
  </queryResult>
</dataOutput>
</view>
```

Code Listing 3. Internal structure of a view

The queryResult references the sqlQuery of Code Listing 2 and is embedded in the auction flow as shown in Code Listing 1. The child element attribute details this reference by specifically selecting the column ParticipantID from the SQL statement's result. So, if the ID of the requesting participant is an element of the returned ParticipantIDs, the data compiled by the dataOutput is returned. In case a participant is eligible to several co-existing views, the auction runtime environment returns the result of all of them. The dataOutput refers to the SQL query highestBid which is sketched in Code Listing 4.

```
SELECT MAX(MaxAmount) AS price FROM OutgoingProduct
WHERE
Timestamp IN (SELECT Timestamp FROM Intention WHERE
StageID=2)
```

Code Listing 4. SQL statement for retrieving the highest submitted bid – highestBid

Besides the outlined view for the seller in an auction, the FPSB example needs a second view revealing the seller's outgoing product to the bidders. A bidder should, for example, be able to see all characteristics of the product to be auctioned.

The DAL approach we present here contrasts known parameterization approaches, which break down the auction design space to give a structured characterization of auction rules (e.g., Wurman *et al.* 2001; Ströbel and Weinhardt 2003). One parameter is, for example, the *order book* which defines the visibility of orders – or offers – to bidders while the auction runs. As Wurman *et al.* (2001) put it: the 'most common choices are to keep the book closed, reveal only the current winning bids or to open the book completely'. This simple example shows the strength as well as the weakness of a parameterization: on the one hand it does a great job at capturing common choices for common parameters but on the other hand it is restricted to these commonalities. There might be any arbitrary number of values for the parameter; but, if one wants to specify an auction with a more sophisticated rule or assign different parameter values to different user roles, parameterization approaches come to a limit. It is, for example, beyond the scope of this parameterization to enable the precise specification of the order book depth and what information (prices, amounts, bidder names, bidder

pseudonyms, etc.) exactly is provided to which participant in what stage.

Parameterization approaches are limited in granularity and innovation to previously identified parameters with their possible values (Lochner and Wellman 2004). On the contrary, DAL provides a flexible language for introducing innovative auction functionality on the fly and in full detail.

Validation. Any stage of an auction can accept intentions contained in offers. Intentions may be checked for acceptance concerning the product they specify, the amount of the product, the associated participant, etc. It is also possible, that no intention at all or any intention is accepted. The *validation* is responsible for checking intentions with respect to required criteria. If all criteria turn out to be fulfilled, the intention is stored in the data basis, otherwise it is discarded. There are zero or more validations in every stage. In Figure 3, validations are depicted as stripes. They are active from the beginning to the end of the respective stage. Code Listing 5 exemplifies the internal structure of a validation in stage 2 of the example auction. All intentions from bidders that specify their incoming product as the outgoing product of the seller's offer are accepted.

```
<validation>
  <equals>
    <queryResult queryName='sellerOutgoingProduct'>
      <attribute name='ProductID'/>
    </queryResult>
    <queryResult queryName=
      'submittedIntentionIncomingProduct'>
      <attribute name='ProductID'/>
    </queryResult>
  </equals>
</validation>
```

Code Listing 5. Internal structure of a validation

The code example refers to two queries and compares their results. If the results are equal the intention is accepted. Otherwise all other validations in the stage are checked for whether they accept the intention. In this example there are no other validations. The former query `sellerOutgoingProduct` refers to a `sqlQuery`. The latter query `submittedIntentionIncomingProduct` refers to a `validationQuery` of the same stage. A `validationQuery` is specified in SQL but does not operate on the auction's data basis, as the newly arriving intention is not yet inserted into this data basis. Instead, the incoming intention is handled as a set of 'virtual tables' by the runtime environment and follows the intention structure outlined in the data model. Besides the exemplified structure, the XML elements of DAL allow for Boolean operations,

comparisons, set operations, constants and simple arithmetic.

Transition. Every stage ends according to certain rules. These rules are represented in *transitions*. There is exactly one transition in every stage. In Figure 3, transitions are depicted as triangles. They are active throughout a whole stage but only affect the auction process when finally ending the respective stage.

For ending a stage a transition can react to newly accepted intentions as well as timer events. An arbitrary number of timers can be employed and dynamically reset on those events within the transition. A soft end like Amazon.com employs (Roth and Ockenfels 2002) can then, for example, be realized by postponing the timer if a new intention is accepted sufficiently close to the prospective end.

Any timer is initialized by the market engineer with either an absolute time or a time span that starts with the beginning of the respective stage. Code Listing 6 exemplifies the simplest version of a timer for a stage which lasts seven days.

```
<transition>
  <timer>
    <timeInterval>
      <days>7</days>
    </timeInterval>
  </timer>
</transition>
```

Code Listing 6. Timer-based transition

A more complex example for a transition reacting on incoming offers is presented in Code Listing 8 in the context of an English auction with soft close.

Agreement generator. Whenever a deal is concluded this is documented with a newly generated agreement in the data basis. In each stage there is either one *agreement generator* or none. In Figure 3, agreement generators are depicted as squares attached to the left edge of the respective stage rectangle. They are active at the very beginning of a stage.

In the FPSB auction, the agreement generator matches exactly two intentions, concretizes them and automatically inserts a corresponding entry in the data basis agreement table. Code Listing 7 outlines the `agreementGenerator` which is executed at the beginning of the FPSB auction's stage 3, after bids were collected in stage 2.

```
<agreementGenerator>
  <agreement>
    <intention>
      <queryResult queryName='highestBidderIntention'>
        <attribute name='IntentionNo'/>
        <attribute name='Timestamp'/>
      </queryResult>
    </intention>
  </agreement>
</agreementGenerator>
```

```

    </queryResult>
  </intention>
</intention>
  <queryResult queryName='sellerIntention'>
    <attribute name='IntentionNo'/>
    <attribute name='Timestamp'/>
  </queryResult>
</intention>
<concretisation>
  <product>
    <queryResult queryName='moneyProduct'>
      <attribute name='ProductID'/>
    </queryResult>
  </product>
  <amount>
    <queryResult queryName='highestBid'>
      <attribute name='price'/>
    </queryResult>
  </amount>
</concretisation>
</agreement>
</agreementGenerator>

```

Code Listing 7. Internal structure of an agreement generator

The presented agreementGenerator utilizes several sqlQueries; their exact specification is omitted for brevity. What is achieved, however, is that the intention of the seller and the highest bidder are combined in an agreement, with all money amounts concretized to the highest bid of all bidders.

Several agreements – for example for implementing a two-sided call market – can be achieved by either concatenating several agreement-tags or by putting a while-loop around the agreement.

Derivation of other auction mechanisms

The previous section outlined several components needed for a FPSB auction. In this section, we demonstrate how auction mechanisms can be derived from other mechanisms in DAL and how complex auction types can be built by recombining existing components. A DAL specification of an auction does not only serve as definition for this specific auction but it is a toolbox for reusing components in a plug-and-play like recombination.

Second-price sealed-bid auction. As the name suggests, a second-price sealed-bid auction differs from a first-price sealed-bid auction in the determination of the price. The winner is determined in the same way, i.e. as the bidder with the highest bid, but the price this winner pays is the second highest submitted bid amount. Accordingly, we can transfer the example auction into a second-price sealed-bid auction – sometimes also referred to as Vickrey auction – by only replacing the query highestBid with secondHighestBid in the

introduced concretization of the agreement generator. The respective query must be added to stage 3.

English auction. An English auction is characterized by iterative offer submission and information feedback for the bidders on the auction status – especially the highest bid amount. Following the component idea of DAL, we merely have to modify a view in the second stage of the introduced FPSB auction: The view presented in Code Listing 3 is modified by replacing the checkRole-tag with an empty one, as everyone is allowed to access the view in the English auction. The iterative offer submission required for an English auction was already allowed by the validation in Code Listing 5.¹ This means that the validation can be adopted without modification as well as any other component plus the auction flow of the FPSB auction defined above. So, by a small modification in a single view, we have generated a new type of auction.

Beyond the hard close of stage 2, as it is defined in the example auction, a soft close is also common in English auctions. This aspect can easily be modelled by replacing the transition of stage 2 in Code Listing 6 by the one given in Code Listing 8.

```

<transition>
  <timer name='timer1'>
    <timeInterval>
      <days>1</days>
    </timeInterval>
  </timer>
  <stoppingRule>
    <condition>
      <lessThan>
        <systemVariable>timer1</systemVariable>
        <plus>
          <systemVariable>currentTime</systemVariable>
          <timeInterval>
            <hours>1</hours>
          </timeInterval>
        </plus>
      </lessThan>
    </condition>
    <action>
      <setTimer timer='timer1'>
        <plus>
          <systemVariable>currentTime</systemVariable>
          <timeInterval>
            <hours>1</hours>
          </timeInterval>
        </plus>
      </setTimer>
    </action>
  </stoppingRule>
</transition>

```

Code Listing 8. Transition for soft end in an English auction

The transition ends a stage if no new offer was accepted for one hour and earliest after one day. Two points are

noteworthy here: the transition ends the stage as soon as any timer fires and the stopping rule is checked each time an intention is added to the data basis. Modelling transitions in that form significantly extends the capabilities of parameterization.

Anglo-Dutch auction. An example for an auction with more than three stages is proposed by Klemperer (1998) for encouraging entry by weak bidders. For auctioning one object, the so called Anglo-Dutch auction is defined as an auction with firstly rising prices in an English auction and then secondly a sealed-bid auction in which only the two high bidders from the English auction are allowed to participate. The highest bidder in the sealed-bid stage wins.

To describe this auction, we draw on four stages. Stage 1 is identical to the first stage of the auctions presented above. Stage 2 is identical to the second stage of the English auction with a soft end. Stage 3 equals the second stage of the FPSB auction, except for the validation that now only accepts offers from the two highest bidders of the previous stage, while the specified prices of the two bidders must be equal or higher than their respective highest bid in stage 2. Stage 4 is equal to the third stage of the first-price sealed-bid auction except for the stageIDs in the query that must both be incremented by one.

TRADING SOFTWARE AGENTS

One goal of DAL is to enable software agents to derive their bidding behaviour at runtime. Before participation, a software agent can request the description of a running auction mechanism and learn the functioning of the mechanism, i.e. derive meaningful actions based on general rules.

To give an intuition on how DAL facilitates this mechanism assessment, we first present how agents can derive valid actions from the DAL description. In a second step we sketch the derivation of reasonable actions as a subset of valid actions.

Valid actions

Valid actions comprise all possible actions of an agent with respect to a given auction mechanism. Agents can interact with a mechanism by sending offers that are processed by the validation component and by requesting views. Beyond the market data examples provided above, views can naturally comprise basic data such as the stage ID, the stage instance ID and the start time of the corresponding stage – all termed control data. Some views merely contain such control data. When granted a view with sufficient control data, an agent gets to know the current stage and time information. Based on this an

agent can then, for example, turn to the validation component of the current stage to check whether a prepared offer will be accepted or not. For the following deliberations we presume that any participant has full access to an additional view with all required control data, such as the current stage ID.

The two issues to be addressed in order to derive valid actions are the period of time a participant is allowed to submit offers and the embodiment of valid offers. An agent does not always know in advance whether his offer will be accepted. We distinguish between two cases:

1. The agent can immediately conclude whether his offer will be accepted by checking his offer for the static constraints specified in the validation. In case of the FPSB auction, for example, a buyer's offer submitted during the first stage is ignored as only offers with money as incoming product are accepted.
2. The consideration of an offer may depend on the market instance data of the running auction. This data may either be accessible to the agent by one of his views or not. In case this information is not provided by any view, the agent is incapable of learning in advance whether his offer will be accepted. He can still submit his offer and may then have the chance to conclude from subsequent views whether his offer was accepted.

In the FPSB auction, for example, the validation component in stage 2 accepts offers only if the product IDs of the buyer's incoming product and the seller's outgoing product match. The validation component was given as an example in Code Listing 5. The buyer agent sees that he must learn the seller's outgoing product in order to compare it to his incoming product. In this case he can see this data in his view, and when the two products match, the agent submits his offer to the auction with guaranteed success of acceptance. If this buyer's view does not provide any information about the seller's outgoing product, the agent can nevertheless submit his offer. Whether it is accepted or not might then be visible in subsequent views. In the end, it is up to the market engineer if he wants the mechanism to provide the required data or not.

Reasonable actions

Within the space of valid actions an agent pursues a certain strategy that he defines as reasonable and that supposedly helps in reaching the goals of his principle. Once an agent has learnt how an auction mechanism works and what he is allowed to do, he can derive reasonable actions on the basis of general prior knowledge.

In an English auction a buying agent could for instance become active whenever he is outbid. An agent

learns he is the highest bidder when his participant ID equals the participant ID in the view showing the highest bid. When the agent is not the highest bidder and designed to still bid, he must reason about the respective price. In the agreement generator he learns that only the highest bid wins the auction and pays its price. In case of ties only the earlier bid is considered. Therefore, it is not reasonable to submit offers lower than or equal to the standing highest bid.

With the indicated agreement generator, it is also not reasonable for an agent to bid more than the price his principle is willing to pay. Within the range from the standing highest bid and his principle's valuation, one reasonable strategy would be immediately bidding a small increment above the current price.

All in all, reasonable actions are the result of analysing a mechanism's description based on general knowledge about market mechanisms plus simple strategy building blocks. The abovementioned strategy, for example, can be applied whenever a pay-as-you-bid mechanism allows for submitting bids with prices higher than the standing highest bid. The two remaining set screws then are the precise increment amount – possibly relative to the standing highest bid – and the point of time at which the agent becomes active again, e.g. for checking the view once more. This demonstrates how reasonable actions for agents can be derived with simple steps.

CONCLUSIONS

Electronic markets have been enriched by the market engineers' wealth of auction inventions as well as the growing popularity and lasting awareness of electronic marketplaces. With the prospering variety of market mechanisms, however, electronic trading has grown increasingly demanding and time-consuming.

As a fundament for the assessment of various protocols, we propose market mechanism specifications according to a descriptive auction language that are made available to any market participant. This suffices for automated analysis of protocols to support humans in their assessment and software agents in autonomous interaction, even with unknown auction formats. We indicated an agent's reasoning for deducing valid and reasonable actions in an auction. The concepts of a DAL market server are implemented prototypically; building agents that are capable of interpreting the entire DAL is, however, up to future work. Furthermore, a great deal of research is necessary for automatically deriving truly reasonable strategies and maybe game-theoretic equilibria.

Similar to existing systems like AuctionBot, AB3D and meet2trade, DAL also supports market engineers in further maturing market mechanisms by providing a market runtime environment for immediate execution and rapid evaluation. Beyond established auctions

considered beforehand, DAL also supports newly designed auctions.

DAL builds on prior work on the issue of coherently defining auctions, namely the auction reference model (Rolli and Eberhart 2005) and the minimal market model (Rolli *et al.* 2004). These ideas are advanced to presenting DAL as a comprehensive language for specifying auction mechanisms. DAL draws on a SQL-subset for accessing a market's data basis and wraps the SQL queries in an XML-based language defining the *auction flow* as well as the pivotal components of every auction mechanism: (1) *views* to present information to bidders, (2) *validations* to determine whether a bid is accepted by the auction or not, (3) *transitions* to determine the end of stages and proceed with the following ones and finally (4) *agreement generators* to transform agents' bids in form of intentions to contracts among the agents.

DAL is less restrictive than pure parameterization approaches as it is not limited to a-priori known parameter sets. However, the approach's expressiveness and flexibility are bought at some complexity. First, specifying an auction from scratch via DAL is more demanding for a market engineer than assigning values to parameters, if he is familiar with the parameters' meaning and interplay. Second, computational complexity within the DAL market server grows with the flexibility. Simple and fast constructs applicable within parameter-based engines have not been adopted or countered, yet. We believe, however, that it is possible to implement computationally demanding processes that occur in some auctions in a procedural, object-oriented, etc. way and to nevertheless describe their working via DAL. This retains the market server's external interface and makes the internal working more efficient. Finally, interpreting DAL and deducing bidding behaviour at runtime is certainly more challenging for a software agent than following a fixed hard-coded strategy. However, we find that any agent can readily be equipped with tailored background knowledge on previously known auction mechanisms and thereby always measure up to any other approach in the performance of a strategy employed – albeit with a more complex interface. With future work we intend to demonstrate that even initially rudimentary heuristics will give DAL an edge over inflexible alternatives when it comes to facing the demanding task of interpreting entirely new mechanisms.

Furthermore, for creating auction descriptions DAL heavily supports reuse of components – starting on the level of single SQL statements and ranging over reusing views, validations and the like to recombining entire stages and auction flows. Therefore, a library of standard components could facilitate a fast 'plug-and-play specification' of standard auction classes and a graphical editor will provide support for creating and handling DAL documents more intuitively.

ACKNOWLEDGEMENTS

This research was partially funded by the German Federal Ministry of Education and Research (BMBF) in the SESAM-project as part of the research program Internet Economics and by the German Research Foundation (DFG) within the scope of the Graduate School ‘Information Management and Market Engineering’ (IME). The authors are responsible for the content of this publication.

Note

1. The validation for an English auction could require every new bid to beat the standing highest one. We presume for our example that the one validation for both the FPSB and the English auction example accepts any bid without price restrictions. Nevertheless, if the new bid does not exceed the high bid, it has no chance of winning the auction.

References

- Anthony, P. and Jennings, N. R. (2003) ‘Developing a Bidding Agent for Multiple Heterogeneous Auctions’, *ACM Transactions on Internet Technology (TOIT)* 3(3): 185–217.
- Cheng, S.-F., Leung, E., Lochner, K. M., O’Malley, K., Reeves, D. M., Schwartzman, L. J. and Wellman, M. P. (2005) ‘Walverine: A Walrasian Trading Agent’, *Decision Support Systems* 39(2): 169–84.
- Engelbrecht-Wiggans, R. (1980) ‘Auctions and Bidding Models: A Survey’, *Management Science* 26(2): 119–42.
- Friedman, D. (1993) ‘The Double Auction Market: A Survey’ in D. Friedman and J. Rust (eds) *The Double Auction Market: Institutions, Theory, and Evidence*, Cambridge: Perseus Publishing.
- Klemperer, P. (1998) ‘Auctions with Almost Common Values’, *European Economic Review* 42(3–5): 757–69.
- Lochner, K. M. and Wellman, M. P. (2004) ‘Rule-Based Specification of Auction Mechanisms’, *Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS’04)*, New York, USA.
- McAfee, P. R. and McMillan, J. (1987) ‘Auctions and Bidding’, *Journal of Economic Literature* 25(2): 699–738.
- Rolli, D. and Eberhart, A. (2005) ‘An Auction Reference Model for Describing and Running Auctions’, 7. *Internationale Tagung Wirtschaftsinformatik*, Bamberg, Germany.
- Rolli, D., Neumann, D. and Weinhardt, C. (2004) ‘A Minimal Market Model in Ephemeral Markets’, *TheFormEMC*, Toledo, Spain.
- Roth, A. E. and Ockenfels, A. (2002) ‘Last-Minute Bidding and the Rules for Ending Second-Price Auctions: Evidence from eBay and Amazon on the Internet’, *American Economic Review* 92(4): 1093–103.
- Sinz, E. J. (1988) ‘Das Strukturierte Entity-Relationship Modell (SERM)’, *Angewandte Informatik* 30(5): 191–202.
- Ströbel, M. and Weinhardt, C. (2003) ‘The Montreal Taxonomy for Electronic Negotiations’, *Journal of Group Decision and Negotiation* 12(2): 143–64.
- Weinhardt, C., Holtmann, C. and Neumann, D. (2003) ‘Market Engineering’, *Wirtschaftsinformatik* 45(6): 635–40.
- Weinhardt, C., van Dinther, C., Kolitz, K., Mäkiö, J. and Weber, I. (2005) ‘meet2trade: A Generic Electronic Trading Platform’, *4th Workshop on e-Business (WEB 2005)*, Las Vegas, USA.
- Wurman, P., Wellman, M. P. and Walsh, W. E. (2001) ‘A Parametrization of the Auction Design Space’, *Games and Economic Behavior* 35(1–2): 304–38.
- Wurman, P., Wellman, M. P. and Walsh, W. E. (2002) ‘Specifying Rules for Electronic Auctions’, *AI Magazine* 23(3): 15–23.