Enabling Automatic Certification of Online Auctions

Wei Bai
Department of Computer Science
University of Liverpool
England, UK
Wei.Bai@liverpool.ac.uk

Emmanuel M. Tadjouddine
Department of Computer Science and Software Engineering
Xi'an Jiaotong-Liverpool University
SIP, Suzhou, China
Emmanuel.Tadjouddine@xjtlu.edu.cn

Yu Guo
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China

We consider the problem of building up trust in a network of online auctions by software agents. This requires agents to have a deeper understanding of auction mechanisms and be able to verify desirable properties of a given mechanism. We have shown how these mechanisms can be formalised as semantic web services in OWL-S, a good enough expressive machine-readable formalism enabling software agents, to discover, invoke, and execute a web service. We have also used abstract interpretation to translate the auction's specifications from OWL-S based on description logic, to COQ, based on typed lambda calculus, in order to enable automatic verification of desirable properties of the auction by the software agents. For this language translation, we have discussed the syntactic transformation as well as the semantics connections between both concrete and abstract domains. This work contributes to the implementation of the vision of agent-mediated e-commerce systems.

1 Introduction

Trust is an important issue in building up markets for rational participants. At the heart of markets lie the mechanism that spells out the rules of engagements for any participant. If the rules lack robustness, then paticipants may exploit the perceived loop holes to their benefit. To illustrate this statement, let us consider the example of the LIBOR (London Inter Bank Offered Rate) scandal, see for example [14]. The LIBOR relies on a simple mechanism wherein leading banks in London report estimates of interest rates that they would be charged if they borrow from other banks; we then drop the four highest and the four lowest rates and then calculate the arithmetic mean of the remaining rates. This mechanism clearly relies upon trust: banks will report their rates truthfully. However, a small change in the LIBOR rate could generate large payments to a bank giving incentives to certain banks to influence the LIBOR rate. The so-called LIBOR scandal arose from a misreporting of interest rates by collusion and manipulation. The question is can we ensure that participants are truthful in their reports? In a much more general setting, can we ensure trust in the market?

In this paper, we consider electronic markets wherein software agents can buy or sell goods through online auction houses. This implies software agents will be engaging financial transactions and entering legally binding contracts on behalf of their owners. We would like to enable agents to check that the auction house is trustworthy before entering it and bid for items on sale. For that purpose, we assume that electronic institutions publish the specifications of their protocols in a high-level language specifying who can bid, in what order, and how the winners and payments are determined. A roaming agent who arrives at a foreign institution can download a protocol and analyze it in order to make a decision about whether or not to participate, and what strategy to use.

Submitted to:
FESCA 2014

To illustrate these requirements, we need to bear in mind that an auction is typically defined by a winner determination algorithm; there are several in the literature, see [6]. This provides the winning criteria and therefore implicitly determines the optimal strategy for participating agents. These winner determination algorithms are designed with desirable properties such as truthful bidding is optimal. If an electronic auction house is using such a protocol, it is not sufficient for the auctioneer to claim that a given bidding strategy is optimal. The fact is that sincerity cannot be assumed in open systems. These kinds of requirements are necessary to fully exploit the potential of e-commerce systems and to eventually achieve the vision of agent mediated e-commerce. In summary, the motivation for this work comes from the belief that verifying certain properties of a trading platform are useful to agents in open e-commerce environments only if: i) their protocols can be published in a machine-readable form; ii) their properties can be automatically verified by a software agent. Our focus is to address these requirements by providing a suitable specification language and associated proof procedures.

The Semantic Web not only enables greater access to content but also to service on the Web. OWL-S [13] ontology is a language to describe Web services. It can be used to describe the properties and capabilities of a Web service in an unambiguous and computer-interpretable form. As a result, we have formalised an online auction as a semantic web service by using OWL-S since this provides machine-understandable specifications for software agents. To enable automatic verification procedures by agents, we present a translation framework that maps OWL-S into COQ specifications so that verification can be carried out as we have shown in our previous work [1] wherein we have implemented a verification procedure using the proof-carrying code paradigm from within COQ. The OWL-S-to-COQ language translation relies upon the abstract interpretation approach [5] allowing us to reason within COQ and then deduce properties for OWL-S. The contributions of this ongoing work are two-fold: i) we have provided a machine readable formalism for online auctions by using OWL-S thus enabling software agents to automatically discover, invoke and execute an online auction; ii) we have shown how to map OWL-S auction into COQ specifications by using abstract interpretation so as to enable automatic verification by relying upon a well-established theorem prover.

2 Background and Model

Generally speaking a *combinatorial auction* [6] is composed of a set I of m items to be sold to n potential buyers. A bid is formulated as a pair $(B(x),b(x))$ in which $B(x) \subseteq I$ is a bundle of items and $b(x) \in R^+$ is the price offer for the items in B. Given a set of k bids, $X = \{(B(x_1),b(x_1)),(B(x_2),b(x_2)),\ldots,(B(x_k),b(x_k))\}$ the *combinatorial auction problem*(CAP) is to find a set $X_0 \subseteq X$ such that $\sum_{x \in X_0} b(x)$ is maximal, subject to the constraints that for all $x_i,x_j \in X_0 :\ B(x_i) \cap B(x_j) = \emptyset$ meaning an item can be found in only one accepted bid. We assume *free disposal* meaning that items may remain unallocated at the end of the auction.

As shown in [16], the CAP is NP-hard implying that approximation algorithms are used to find a near-optimal solutions or restrictions are imposed in order to find tractable instances of the CAP [21] wherein polynomial time algorithms can be found for a restricted class of combinatorial auctions. A combinatorial auction can be *sub-additive* (for all bundles $B_i,B_j \subseteq I$ such that $B_i \cap B_j = \emptyset$, the price offer for $B_i \cup B_j$ is less than or equals to the sum of the price offers for B_i and B_j) or *super-additive* (for all $B_i,B_j \subseteq I$ such that $B_i \cap B_j = \emptyset$, the price offer for $B_i \cup B_j$ is greater than or equals to the sum of the price offers for B_i and B_j).

Game theory *mechanism*, see for example [6], is usually used to describe auctions, thus providing decision procedures that determine the set of winners for the auction according to some desired objective.

An objective may be that the mechanism should maximise the *social welfare*, which can be for example the sum of all agents' utilities in the auction. Such a mechanism is termed *efficient*. For open multi-agent systems, another desirable property for the mechanism designer can be *strategyproofness* (truth telling is a dominant strategy for all agents). A mechanism that is strategyproof has a dominant strategy equilibrium.

A well known class of mechanisms that is efficient and strategyproof is the Vickrey-Clarke-Groves (VCG), see for example [6]. The VCG mechanism is performed by finding (i) the allocation that maximises the social welfare and (ii) a pricing rule allowing each winner to benefit from a discount according to his contribution to the overall value for the auction. To formalise the VCG mechanism, let us introduce the following notations:

- $\mathscr{X}$ is the set possible allocations
- $v_i(x)$ is the true valuation of $x \in \mathscr{X}$ for bidder i
- $b_i(x)$ is the bidding value of $x \in \mathscr{X}$ for bidder i
- $x^* \in \text{argmax}_{x\in\mathscr{X}} \sum_{i=1}^{n} b_i(x)$ is the optimal allocation for the submitted bids.
- $x^*_{-i} \in \text{argmax}_{x\in\mathscr{X}} \sum_{j\neq i}^{n} b_j(x)$ is the optimal allocation if agent i were not to bid.
- u_i is the utility function for bidder i.

The VCG payment p_i for bidder i is defined as

$$\begin{aligned} p_i &= b_i(x^*) - \left(\sum_{j=1}^{n} b_j(x^*) - \sum_{j=1,\, j\neq i}^{n} b_j(x^*_{-i})\right) \\ &= \sum_{j=1,\, j\neq i}^{n} b_j(x^*_{-i}) - \sum_{j=1,\, j\neq i}^{n} b_j(x^*), \end{aligned} \tag{1}$$

and then, the utility u_i for agent i is a quasi-linear function of its valuation v_i for the received bundle and payment p_i.

$$u_i(v_i, p_i) = v_i - p_i. \tag{2}$$

Let p_i^* and p_i be the payments for agent i when it bids its true valuation v_i and any number b_i respectively. Note p_i, p_i^* are functions of b_{-i}. The strategyproofness of the mechanism amounts to the following verification:

$$\forall i,\ \forall v_i,\ \forall b_i\ u_i(v_i, p_i^*(b_{-i})) \geq u_i(v_i, p_i(b_{-i})). \tag{3}$$

In the equation (1), the quantity $\sum_{j=1,\, j\neq i}^{n} b_j(x^*_{-i})$ is called the *Clark tax*. Another interesting property of this VCG mechanism is that it is *weakly budget balanced* [4] meaning the sum of all payments is greater than or equal to zero. For the VCG properties (e.g. strategyproof) to hold, the auctioneer must solve $n+1$ hard combinatorial optimisation problems (the optimal allocation in the presence of all bidders followed by n optimal allocations with each bidder removed) exactly.

Single item auctions are a particular case of combinatorial auctions wherein one item is sold at a time. Common single item auctions are the English, Dutch or Vickrey auctions, see for example [6]. For the Vickrey auction, the payment for agent i is defined as:

$$\begin{aligned} p_i &= b_i(x^*) - \left(\sum_{j=1}^{n} b_j(x^*) - \sum_{j=1,\, j\neq i}^{n} b_j(x^*_{-i})\right) \\ &= \sum_{j=1,\, j\neq i}^{n} b_j(x^*_{-i}) \ = \ \text{the second highest bid.} \end{aligned} \tag{4}$$

For the English auction, the payment for agent i is defined as:

$$p_i \ = b_i(x^*) \ = \ \text{the highest bid.} \tag{5}$$

The utilities of the Vickrey and English auctions are defined as in equation (2).

We assume that an online auction will run over a fixed time period T and that the seller may have a reserve revenue under which items cannot be sold. This reserve revenue can typically be the reserve price for single item auctions. After a bid, the seller waits for some time before accepting the bid if it yiels a revenue that is greater or equal to the reserve one or waits for the expiry time before deciding whether to reject or accept the bid. In this work, we aim at describing online auctions by using the semantic web, e.g., the OWL-S specification language and then translate the resulting description into COQ specifications in order to carry out the verification of desirable properties.

3 OWL-S Specifications

Online auctions of software agents is a good application wherein data can be processed by automated reasoning tools. Logic-based languages are useful tools to model and reason about systems. They allow us to specify behavioral requirements of components of a system and formulate desirable properties for an individual component or the entire system. The semantic web enables us to describe and reason about web services by using *ontologies*. Ontologies permit us to describe the semantics of terms representing an area of knowledge. OWL, a W3C standard, is a description logic-based language that enables us to describe ontologies by using basic constructs such as concept definitions and relations between them. It has been used in a wide range of areas including biology, medicine, or aerospace [7]. In this work, we view an auction as a web service with enough logical attachments enabling a software agent to understand the auction rules and to carry out verifications of claimed properties. Logic-based languages are usually chosen for their *expressivity* or on the fact that their underlying logic is *sound*, *complete* or *decidable*. Expressivity provides us with powerful constructs to describe things that may not be otherwise expressed. Soundness ensures that if a property ϕ can be deduced from a system (a set of statements) Γ ($\Gamma \vdash \phi$), then ϕ is true as long as Γ is ($\Gamma \models \phi$). Completeness states that any true statement can be established by proof steps in the logic's calculus. Formally $\Gamma \models \phi$ implies $\Gamma \vdash \phi$. A logic is decidable if for any statement we can construct an algorithm that decides if it is true or false.

3.1 Auction Ontology

Ontology can be used to formally describe the semantics of terms representing an area of knowledge and give explicit meaning to the information. This allows for automated reasoning, semantic search and knowledge management in a specific area of knowledge. The ontology of auction domain contains the following constructs: classes, relations, axioms, individuals and assertions. We have used the ontology language OWL-S, which is used to describe Semantic Web Services within the OWL-based framework in order to build up the auction ontology.

Figure 1 displays the top level concepts of our online auction ontology. The top-level classes are `Agent` and `AuctionService`. The class `Winner` is a subclass of the class `Agent`. The relations between classes are defined as properties. There are two kinds of properties: object properties and datatype properties. The object properties link individuals to individuals while the datatype properties link individuals to data values. In the following section, we show how to encode this ontology within the OWL-S language.

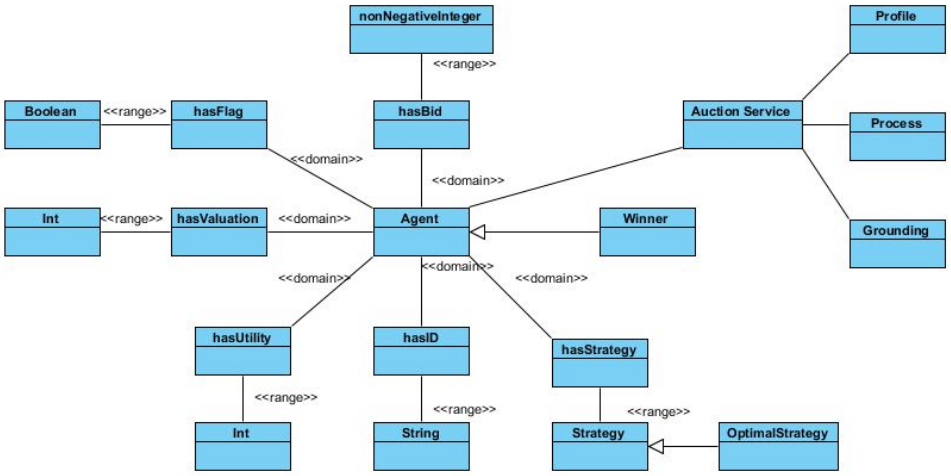
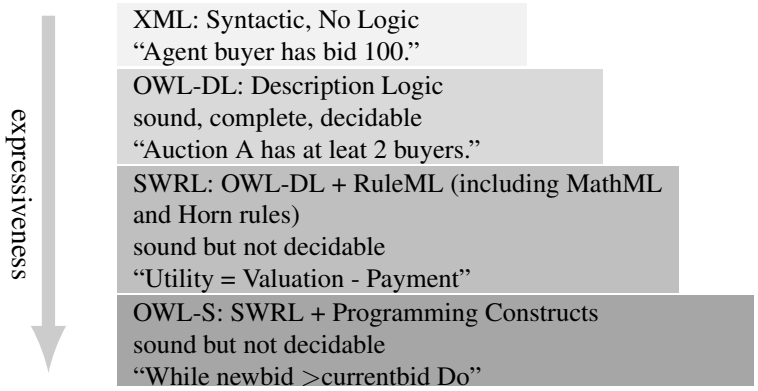


Figure 1: Ontology for an auction service

3.2 The OWL-S Description Language

In this section, we argue that we can describe online auction houses as web services by using the OWL-S language, which provides us with a machine readable formalism and logical reasoning capabilities for software agents. We will start by showing that OWL-S is expressive enough to enable us to describe online auction mechanisms.



To describe online auctions, we may ask why not XML (Extensible Markup Language) as a description language for this task. XML provide a syntactic approach but no logical base for reasoning. The meaning of the relationships between XML elements can not be encoded. A language that builds upon XML and allows for reasoning is the OWL-DL (Web Ontology Language), which is based on DL (Description Logic). Description logics are a family of logics that are decidable fragments of first-order logic. OWL-DL is sound, complete, and decidable but with limited expressivity. For example, we cannot express the statement that 'an agent's utility is its valuation minus its payment'. To extend OWL-DL, the SWRL (Semantic Web Rule Language) combines OWL-DL with RuleML that includes among others, MathML and Horn rules. As a result, SWRL is more expressive than OWL-DL but SWRL is not decidable [10]. However, in SWRL, we cannot express the statement that 'while newbid >currentbid do'. Furthermore, auction mechanisms can be viewed as functions with inputs, outputs, preconditions or postconditions. They may contain complex programming constructs such as branching or iterations.

OWL-S enables us to describe online auctions as Web Services.

An OWL-S description is mainly composed of a service *profile* for advertising and discovering services; a *process* model, which describes the operation of a service; and the *grounding*, which specifies how to access a service. In our case, the process contains information about inputs, outputs, and a natural language description of the auction, e.g., this is a Vickrey auction. The grounding contains information on the service location so that an agent can run the service by using the OWL-S API. The process model is described as follows:

```
define composite process Auction
(inputs: (...)
outputs: (...)
preconditions: (...)
results: (...)
)
{ // Process's Body
WinDetermAlgo(...);
PayeAndUtil(...) }
```

```
<process:CompositeProcess rdf:ID="Auction">
<process:hasInput> ...
<process:hasOutput>
<process:hasPrecondition> ...
<process:hasResult> ...
...
<process:AtomicProcess rdf:ID="WinDetermAlgo"> ...
<process:AtomicProcess rdf:ID="PayeAndUtil"> ...
</process:CompositeProcess>
```

The auction process model is basically composed of inputs, outputs, preconditions, results and a composition of two processes, which are the winner determination algorithm `WinDetermAlgo` and `PayeAndUtil` that calculates the payments as well as the utilities for the buyer agents. In OWL-S, we can also express auction properties such as the one defined by Equation (3) by using composite processes to define utility calculation as a kind of function and `perform` operations. Note that OWL-S allows to express the forall operation in a first-order logic formula. We omit the OWL-S code for Equation (3) for clarity and lack of space.

3.3 Implementation Issues

We have implemented this agent-mediated semantic web service in JADE (Java Agent DEvelopment) framework [2] by using a Yellow Pages service. This permits seller agents to publish their services, so that buyer agents can find and exploit them. All the information of the buyer agents, such as bid and valuation of each agent, are translated into an OWL file, which is imported by the OWL-S service file containing the auction mechanism. We run this service using the OWL-S API to ensure the auction runs as expected. Eventually, the seller agent sends the result to each participant. The ACL language [2] is used for communicating messages between the agents.

4 From OWL-S to COQ Specifications

In Section 3, we have shown how online auction mechanisms can be described using the OWL-S description language in order to have machine-understandable protocols by software agents. On the other hand, we have illustrated that auction mechanisms can be specified within COQ so as to develop machine-checkable proofs of desirable mechanism properties. The verification approach relies upon the proof-carrying code approach to enable the agents to gain confidence at the auction house by automatically verifying desirable properties. To bridge the gap between these two processes, we propose to transform OWL-S into COQ specifications by

1. devise an interpretation that maps the OWL-S semantics into COQ's so as verified properties by COQ can be deduced in the OWL-S domain.
2. translate OWL-S into a tailored subset of COQ specifications so that an OWL-S program or logical formula can be transformed into a COQ one in an automated fashion.

To carry out the above two tasks, we rely upon abstract interpretation, see for example [5]. The OWL-S specifications provide the following: (i) a semantic domain $\mathscr{D}$ of states formed by vocabularies in terms of classes, subclasses, and properties; (ii) for each OWL-S program, an operational semantics $[\![]\!] \subseteq \mathscr{D} \times \mathscr{D}$ with some initial domain $\mathscr{D}^0$, (iii) for a given program p, the collecting semantics $[\![p]\!]$ is defined as the set of reachable states of $\mathscr{D}$ starting by an initial state in $\mathscr{D}^0$. An abstraction $\alpha : (2^{\mathscr{D}}, \subseteq, \cap, \cup) \rightarrow (\mathscr{D}^\sharp, \subseteq^\sharp, \cap^\sharp, \cup^\sharp)$ wherein $\mathscr{D}^\sharp$ is an abstract domain with a lattice structure enables us to map concrete elements into abstract ones. The partial orders $\subseteq$ and $\subseteq^\sharp$ model the precision of the concrete and abstract elements in $2^{\mathscr{D}}$ and $\mathscr{D}^\sharp$ respectively. To define the abstract semantics in the abstract domain, we use a concretisation function $\gamma : (\mathscr{D}^\sharp, \subseteq^\sharp, \cap^\sharp, \cup^\sharp) \rightarrow (2^{\mathscr{D}}, \subseteq, \cap, \cup)$ such that:

$$\forall p\ \alpha(p) \subseteq^\sharp p^\sharp \Rightarrow [\![p]\!] \subseteq \gamma(p^\sharp).$$

The abstraction α can be semantics-preserving if it does not use approximations in the abstract domain. If approximations are used then, conservative analyses are usually used. In this case, it is required that α be at least a *sound* abstraction so as to ensure that for a given program and a property f, $f^\sharp = \alpha(f) = \text{ true } \Rightarrow f = \text{ true}$. If $f^\sharp$ is false in the abstract domain, then we pick the counter-example found in the abstract domain, translate it into the concrete domain and simulate it to discover if it is indeed a counter-example of the concrete program. If this is the case, then f does not hold, see [9].

4.1 Translating Specifications

We have used classical set operations to define OWL descriptions such as classes, properties, individuals and data values within COQ. The following table lists part of the mapping; more details are available upon request.

OWL descriptions	COQ equivalent definitions
Individual	`Variable Individual : Set.`
Class	`Definition Class := set Individual.`
ObjectProperty	`Definition ObjectProperty :=` `  set (prod Individual Individual).`
DatatypeProperty	`Definition DataProperty :=` `  set (prod Individual Value).`

SomeValuesFrom()	`Definition someValuesFrom (op: ObjectProperty)`
	`  (c: Class) : Class :=`
	`  fun i =>`
	`  exists y, set_in (i, y) op /\ (y <- c).`

AllValuesFrom()	`Definition allValuesFrom (op: ObjectProperty)`
	`  (c: Class) : Class :=`
	`  fun i =>`
	`  forall y, set_in (i, y) op -> (y <- c).`

By mapping the OWL-S constructs into COQ equivalent definitions, we can transform an OWL-S specification into a COQ counterpart as illustrated below.

```
<Class IRI="#Agent"/>
   <Class IRI="#Agent"/>
   <NamedIndividual IRI="#agent_1"/>
   <Class IRI="#Agent"/>
   <NamedIndividual IRI="#agent_2"/>
```

```
Inductive Individual : Set :=
  | agent (n: nat).
Definition Agent : Class :=
  set_add (agent _2)
     (set_add (agent _1)empty_class).
```

In the OWL-S code on the left, we define a class `Agent` and two individuals `Agent_1` and `Agent_2` that belongs to the class `Agent`. On the right, we show what will be the result of translating the OWL-S code fragment into COQ. Such a translation will rely on compiler technology.

4.2 Verifying Desirable Properties

In previous work [1], we have relied upon the Proof-Carrying Code (PCC) ideas since it allows us to shift the burden of proof from the buyer agent to the auctioneer who can spend time to prove a claimed property once for all so that it can be checked by a software agent willing to join the auction house. The certification procedure works as follows. The buyer agent arriving at the auction house can download its specification and the claimed proof of a desirable property. Then, the buyer installs the proof checker, which is a standalone verifier for COQ proofs. After the proof checker is installed to the consumer side, the buyer can now perform all verifications of claimed properties of the auction before deciding whether to join and with which bidding strategy.

COQ [22] is an interactive theorem prover based on the calculus of inductive constructions allowing definitions of data types, predicates, and functions. It also enables us to express mathematical theorems and to interactively develop proofs that are checked from within the system. We have used COQ because it has been developed for more than twenty years and is widely used for formal proof development in a variety of contexts related to software and hardware correctness and safety. For example, COQ was used to develop and certify a compiler [11]. A fully computer-checked proof of the Four Colour Theorem was created in [8]. In [23], a COQ-formalised proof that all non-cooperative and sequential games have a Nash equilibrium point is presented. It turns out that COQ is a good example of combination of logic and computation as it allows for the formalisation of different types of logic (e.g., first order logic, etc.) while providing the possibility of defining functions, which are the cornerstone of computation.

A well-defined auction mechanism can be viewed as a function that maps a set of typed agents into outcomes characterised by utilities usually defined as pseudo-linear equations, see Section 2. Not only, we need to specify rules and properties but we also need to carry out some calculations. The constructive approach provided by COQ offers possibilities to describe auctions along with desirable properties and prove them. In [1], we have developed the COQ proof of the well-known statement that bidding its true valuation is the dominant strategy for each agent in a Vickrey or English auction. Another important property that can be checked is that the auction is a well-defined function and that it does implement its specifications through certified code generation [3]. More challenging properties to be checked might be that the auction mechanism is collusion-free or that it is free from fictitious bidding. In this work, we focus on the mapping of auction specifications from OWL-S to COQ so as to enable the verification in a more dedicated and powerful proof development tool.

5 Related Work

There are several related works about auction ontologies. In [25], a framework called TAGA is used to simulate the automated trading in dynamic markets. TAGA uses semantic web language to specify and publish underlying common ontologies. However, the specification of auction services in this paper does not describe the explicit process and rules of different auctions. In [20], a shared negotiation ontology is used to help agents negotiate with each other with a possible application to simulate online auction competition. The simulation was not a real implementation but just an illustration. In [17], the authors integrate semantic web technologies into agent architecture to represent the knowledge and behavior of agent. Although, this paper is mainly about distributed knowledge management, the presented approach can help us building up an agent mediated and automatically processed online auction service.

The idea of software agents automatically checking desirable properties has been investigated in [19] by using a model checking approach. The computational complexity of such costly verification procedures are investigated in [18]. More recently, a proof-carrying code approach relying on proof develop-

ment tools to enable automatic certification of auction mechanisms has been investigated in [1, 3]. Our work is aimed at bridging the gap between the need for a machine-readable formalism to specify online auction mechanisms and the ability of software agents to check possibly complex auction properties.

In the context of specifications translation, the work reported in [12] has illustrated a case of transformation from OWL to Z specifications. The soundness of this mapping is shown using an algebraic institution morphism by viewing the underlying logics of OWL and Z as institutions. In [15], OWL-S was mapped into Frame logic for the use of first order logic based model checking to verify certain properties of semantic web service systems. In [24], a matching approach was used to select appropriate component software in a computer program translation scheme by inspecting the component software specifications. In our work, we define a syntactic translation scheme followed by a semantics interpretation based on the abstract interpretation paradigm.

6 Concluding Remarks

In this paper, we have considered the problem of trust in a network of online auctions by software agents. We have focused on the specifications of auction mechanisms required to be machine-understandable and have proposed an OWL-S formalisation of the mechanisms. We have discussed how OWL-S is expressive enough for the description of these mechanisms. To enable automatic verification of auction properties by software agents, we have relied upon the well-established proof development COQ based on lambda calculus by translating the auction specifications from OWL-S to COQ. For this translation, we have illustrated the syntactic transformation and have used abstract interpretation to connect the concrete semantics of the OWL-S domain into the semantics of COQ.

Future work includes the followings. We first need to formally establish the soundness (possibly the completeness) of the translation of auction specifications from OWL-S to COQ. We will then build up the infrastructure that connects the agents simulation tool JADE to COQ in an automated fashion. In JADE, we should be able to simulate two or three auction services and a number of software agents capable of understanding those auctions thanks to their OWL-S description, translate their specifications from OWL-S to COQ so that proofs of desirable properties of those mechanisms can be developed from within COQ, and then finally, enable a software agent to connect to COQ in order to certify a given property by using the proof-carrying code as shown in [1]. Note that the translation from OWL-S to COQ relies upon compiler construction technology. Such an infrastructure will enable us to simulate *artificial markets* (since they are mediated by software agents) that will shed some light on real markets by making assumptions and analysing their effects on the specified artificial societies.

References

[1] Wei Bai, Emmanuel M. Tadjouddine, Terry Payne & Steven Guan (2013): *A Proof-Carrying Code Approach to Certificate Auction Mechanisms*. In: *The 10th International Symposium on Formal Aspects of Component Software*.

[2] Fabio Bellifemine, Federico Bergenti, Giovanni Caire & Agostino Poggi (2005): *JADE – A Java Agent DEvelopment Framework*. In: *Multi-Agent Programming*, Springer, pp. 125–147.

[3] Marco B Caminati, Manfred Kerber, Christoph Lange & Colin Rowat (2013): *Proving soundness of combinatorial Vickrey auctions and generating verified executable code*. *arXiv preprint arXiv:1308.1779*.

[4] Edward H Clarke (1971): *Multipart pricing of public goods*. *Public choice* 11(1), pp. 17–33.

[5] P. Cousot & R. Cousot (2004): *Basic concepts of abstract interpretation*. Building the Information Society, pp. 359–366.

[6] Peter Cramton, Yoav Shoham & Richard Steinberg (2006): *Combinatorial auctions*.

[7] A-S Dadzie, R Bhagdev, A Chakravarthy, S Chapman, J Iria, V Lanfranchi, J Magalhães, D Petrelli & F Ciravegna (2009): *Applying semantic web technologies to knowledge sharing in aerospace engineering*. Journal of Intelligent Manufacturing 20(5), pp. 611–623.

[8] G. Gonthier (2008): *The four colour theorem: Engineering of a formal proof*. Computer Mathematics, pp. 333–333.

[9] Bhargav S Gulavani & Sriram K Rajamani (2006): *Counterexample driven refinement for abstract interpretation*. In: Tools and Algorithms for the Construction and Analysis of Systems, Springer, pp. 474–488.

[10] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean et al. (2004): *SWRL: A semantic web rule language combining OWL and RuleML*. W3C Member submission 21, p. 79.

[11] X. Leroy (2009): *Formal verification of a realistic compiler*. Communications of the ACM 52(7), pp. 107–115.

[12] Dorel Lucanu, Yuan-Fang Li & Jin Song Dong (2005): *Institution Morphisms for Relating OWL and Z*. In: SEKE, pp. 286–291.

[13] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne et al. (2004): *OWL-S: Semantic markup for web services*. W3C member submission 22, pp. 2007–04.

[14] James McAndrews, Asani Sarkar & Zhenyu Wang (2008): *The effect of the term auction facility on the london inter-bank offered rate*. Technical Report, Staff Report, Federal Reserve Bank of New York.

[15] Huaikou Miao, Tao He & Liping Li (2009): *Formal semantics of OWL-S with F-logic*. In: Computer and Information Science 2009, Springer, pp. 105–117.

[16] Michael H Rothkopf, Aleksandar Pekeč & Ronald M Harstad (1998): *Computationally manageable combinational auctions*. Management science 44(8), pp. 1131–1147.

[17] Julien Subercaze & Pierre Maret (2011): *Programming Semantic Agent for Distributed Knowledge Management*. In: Semantic Agent Systems, Springer, pp. 47–65.

[18] Emmanuel M. Tadjouddine (2011): *Computational Complexity of Some Intelligent Computing Systems*. International Journal of Intelligent Computing and Cybernetics 4(2), pp. 144 – 159.

[19] Emmanuel M Tadjouddine, Frank Guerin & Wamberto Vasconcelos (2009): *Abstracting and Verifying Strategy-Proofness for Auction Mechanisms*. In: Declarative Agent Languages and Technologies VI, Springer, pp. 197–214.

[20] Valentina Tamma, Michael Wooldridge, Ian Blacoe & Ian Dickinson (2002): *An ontology based approach to automated negotiation*. In: Agent-Mediated Electronic Commerce IV. Designing Mechanisms and Systems, Springer, pp. 219–237.

[21] Moshe Tennenholtz (2000): *Some tractable combinatorial auctions*. In: AAAI/IAAI, pp. 98–103.

[22] The Coq Development Team (2012): *The Coq proof assistant reference manual: Version 8.4*. `http://coq.inria.fr`.

[23] R. Vestergaard (2006): *A constructive approach to sequential Nash equilibria*. Information Processing Letters 97(2), pp. 46–51.

[24] Amy Moormann Zaremski & Jeannette M Wing (1997): *Specification matching of software components*. ACM Transactions on Software Engineering and Methodology (TOSEM) 6(4), pp. 333–369.

[25] Youyong Zou, Tim Finin, Li Ding, Harry Chen & Rong Pan (2003): *Using semantic web technology in multi-agent systems: a case study in the TAGA trading agent environment*. In: Proceedings of the 5th international conference on Electronic commerce, ACM, pp. 95–101.