University of Osnabrück

Masters Thesis

A Framework for Market-based Coordination in Multi-Agent Systems

Author:	Arnim Bleier
First Supervisor:	PhD. Artem Katasonov
Second Supervisor:	Prof. Claudia Pahl-Wostl
Date of Submission:	30.09.2008

Declaration

I hereby declare that this thesis is my own work and effort. Where other sources of information have been used, they have been acknowledged.

Jyväskylä, Finland

30.09.2008 …….

Signature

Abstract

Although the concept of Agents has many advantages when it comes to engineering complex systems, the downside is that their organization is difficult to be specified at the time of design. Literature sketches two major directions for the search of a solution: centralized top-down approaches and (economic) bottom-up mechanisms. Especially the economic direction is not studied much by the scientific community, yet. This thesis describes the theory of Market-based Coordination and presents a prototype implementation. The characteristic features of this thesis are, the policy guided decision method (i.e. the commitment of agents to trades is regulated upon general rules), as well as the realization of domain independency, ensuring interoperability.

Zusammenfassung

Das Konzept von Softwareagenten hat viele Vorteile wenn es darum geht komplexe Systeme aufzubauen. Die Kehrseite ist, daß das Design der Organisation nur schwer im Vornherein festzulegen is. In der Literatur können zwei Strömungen unterschieden werden, die als Anssatz für die Lösung dienen können: zentralisierte Top-down-Ansätze und (ökonomische) Bottom-up-Mechanismen. Insbesondere die ökonomische Richtung ist bisher nur wenig untersucht worden. Diese Arbeit beschreibt die Theorie von Market-based-Koordination und präsentiert darauf aufbauend deren Implementierung in einem Prototyp. Die Hauptmerkmale dieser Arbeit sind, daß Agenten Entscheidungen aufgrund von Policies treffen (d.h. ob ein Agent einen Handel eingeht ist bestimmt durch allgemeine Regeln), sowie die Realisierung von Domain-Unabhängigkeit zur Gerwährleistung von Interoperabilität.

List of Acronyms

ACC	Agent Communication Channel
AMS	Agent Management System
DAML	DARPA Agent Markup Language
DF	Directory Facilitator
FIPA	Foundation of Intelligent Physical Agents
N3	Notation3
MAS	Multi-Agent-Systems
OWL	Web Ontology Language
OWL DL	Web Ontology Language Description Logic
KB	Knowledge Base
RAB	Reusable Atomic Behaviour
RDF	Resource Description Framework
RDFS	RDF Schema
RuleML	Rule Markup Language
S-APL	Semantic Agent Programming Language
SPARQL	Protocol and RDF Query Language
SWRL	Semantic Web Rule Language
URI	Uniform Resource Identifier
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	extensible Markup Language

List of Figures

Table of Contents

1 Introduction

In many domains the concept of Multi-Agent Systems has proved to be useful. A group of agents allows making effective (re-) use of specialists designed for a single task (e.g., supplying weather information, controlling servos, providing a human user interface), instead of having a monolithic program which is specialized at no task and hard to adapt to new domains of application.

However, the organisation of a Multi-Agent System is difficult to be specified at the time of design in the face of a dynamic environment. The challenge is to coordinate all of these agents to follow a single, global purpose. One research stream is to describe a system of multiple collaborating agents as one single agent that has a variety of degrees of freedom. The issue hereby is that this only relocates the question. Furthermore, such an approach demands that all information is transmitted to a central instance, resulting in a highly vulnerable system. Thus, the whole system is disabled if the central coordinating unit fails.

Decentralised approaches address these drawbacks. The assumption is that each agent works essentially independent, acting on locally accessible information. Agents may coordinate with other agents to decompose a task into subtasks or to achieve something that cannot be achieved by a single agent. The benefit of such an approach is that only little communication is required, since the agents only transmit information to their associates. Also, the system is less vulnerable, since the entire functioning no longer relies on a single coordinating unit. This works best for tasks that can be split into largely independent subtasks, or tasks for which the desired performance of the system results upon the aggregation of individual actions. Although some Multi-Agent Systems inspired by social analogues have been reported (e.g. Gaertner, 2007), they are limited to operate on simple problems only. The difficulty is to determine what particular individual actions produce the desired emergent behaviour, and to adopt these systems to varying goals.

In most cases agents solve problems of limited complexity. These agents are designed to perform tasks such as fetching information from a database, controlling a device, or processing data from a sensor. To use an economically inspired system for coordinating such agents seems natural. The field of economics is mainly concerned with investigating a population of specialized (human) agents creating services and goods. In the last century decentralized as well as centralized approaches had been tested on a large scale. The Planned economy uses a centralized method in which the state or government coordinates the production. However, it has been argued that such a central organized system is barley able to use all available information, hard to be optimized, and inflexible to adapt to new requirements. In addition to these back draws the most fatal one is the decupling of action and accountability at the individual level resulting in a very low incentive to achieve more than the plan demands.

In a Multi-Agent System inspired by Market economy agents coordinate with each other to solve problems in a decentralized manner. The agents commit trades as they fit and because of the fact that the participants in the market act only to increase their own utility; the invisible hand lets a highly productive system emerge. It is assumed that the agents themselves have the most appropriate knowledge about their needs and can search for their satisfaction. The principle of the unity of action and accountability ensures that agents harvest the benefits of their good decisions as well as their bad decisions. Agents compete with each other to consume services at the lowest possible pricing or cooperate to accomplish something what one alone would not be able to do. However, independent of the form of interaction agents act only in the pursuit of their utility. On the following pages, Adam Smith's invisible hand touches Multi-Agent Systems. Roughly the structure is divided into three main blocks. In the first block (chapter 2-4) theoretical concepts are investigated. In the second block (chapter 5-7) concepts needed for the third block are introduced. In the last block (chapter 8-11) the market mechanism is constructed step by step.

2 Preliminary Considerations

This initial section has the purpose of making the reader familiar with the requirements a Market-based Multi-Agent System faces. Furthermore, it gives an overview on related work.

2.1 A first Enquiry of Market-based Coordination

In recent years an increasing need to cope with ongoing changes and disturbances in manufacturing systems can be recognized. Consequently constant adaptation and high flexibility are needed for their control (Brussel, 1998). Multi-Agent Systems promise to cope with these challenges successfully.

The basic building blocks of Multi-Agent System (agents) are citizens of two worlds; having an informational component and may control a physical component / resource. The informational component contains information about the state of the world (called Knowledge Base), has rules that allow the operation on its KB, and manages the physical resource. The primary focus in this thesis is on this informational component.

Agents are considered to be autonomous and cooperating (Sierra, 1998). The concept of autonomy views an agent as capable of fulfilling its goals. However, an agent is not isolated but just a part of a system; fostering cooperation between agents is curricular to allow the system to fulfil its purpose. The Market-based approach to coordination is a natural response to Gou Luh and Kyoya (1998), pointing out the importance of localized information and decision-making while maintaining cooperation with other agents.

Terminological coordination refers to the exchange of information for the purpose of joint acting; thus, cooperation. From the perspective of an agent a coordination protocol is needed to transport information from one agent to another one. From the perspective of the system a coordination mechanism is needed to allow cooperation - to improve the overall state of the system. Information-sharing and decision-making capabilities are required to enable cooperation between agents.

In the economic tradition cooperation is the mutual assistance between egoists (Adelsberger, 2000). Whereby egoists are in the case of the Multi-Agent Systems agents trying to achieve their goal as good as possible, i.e. are Utility Maximizing agents. Viewing cooperation as assistance among Utility Maximizing agents leads to the efficiency postulate: Utility Maximizing agents cooperate only if doing so increases the utility of booth agents (Oberender, 2004 p. 8). Consequently, equilibrium analysis is one aspect in the development of algorithms for the computation of efficient outcomes in cooperative situations. The Market-based approach to cooperation allocates resources to agents by means of benefits and costs, calculated by the individual agents. Given a certain good each agent has to decide on its own which opportunity costs it is willing to pay and act accordingly; that is, trying to buy the good it is interested in. Direct exchange of one good against another one is not always possible. Consequently, the presence of a meaningful currency that resembles the scarcity of resources is required (Ygge, 1996). Furthermore, each agent needs mapping from the goods to a utility valuation, called Utility Function; that assigns each good to a value.

In a cooperative situation an agent providing resources, services or goods is called supplier; an agent in demand for it is called consumer. Individual agents are not necessarily designated a priori as supplier or consumers; an agent can act on both sides, depending on the situation. The allocation of goods from the suppliers to the consumers is pareto-efficient, if no agent is harmed and at least one agent is better off in comparison to the situation without the allocation (Oberender, 2004 p. 41). Since this restricts market transactions to those that are mutual beneficial the supplier of a good must receive a compensation that is at least as high as the costs for providing the good; and the buyer cannot be forced to provide a higher compensation than the good it receives is worth for it. The compensation takes place in terms of money; its amount is known as price.

Up to yet, the necessity to trade goods against money between agents supplying goods and agents demanding goods has been recognized. If the market mechanism is working with respect to the information flow, all agents having a demand would ask all agents rotationally satisfying that demand

about the pricing in order to optimize the decision about a trade partner. Consuming agents have an interest to purchase goods from the cheapest provider; and providers have an interest to sell goods to agents that are most in need of them. This leads to an individual decision-making based on a kind of negotiation between all potential buyer-seller pairs. The coordination mechanism of auctioning is required to structure the negotiation (McAfee, 1987).

2.2 Related Work

The Market-based coordination of Multi-Agent Systems is related to many fields of research. In this paragraph various approaches related to the approach taken in this thesis are examined.

Much of the work on negotiations between software agents can be traced back to the Contract Net (Smith, 1980). Smith's work is a communication protocol for distributed problem solving. The aim of Contract Net protocol is to enable opportunistic, adaptive task allocation by announcing tasks, placing bids, and awarding contracts. A manager agent can offer a task to other agents, playing a contractor role, which can in the next step submit bids based on their capabilities to execute the task. In a further step the manager awards the task to one of the bidders, and the task gets allocated to the winner, i.e. contractor. In the Contract Net, agents are not designed statically to be either managers or contractors, but these are roles that agents take on dynamically depending on the situation. Since in the original proposal no (internode-) language for the description of tasks had been specified, the approach remains rather unspecific and limited. Related work includes such as the Enterprise system that allocates tasks by means of negotiation mechanisms as well as protocols supporting coalition formation among agents (Malone, 1988). A more recent paper addressing the issue of language for the description of tasks in the Contract Net have been put forward by Sandholm and Lesser (1995). Protocols of the Contract Net family are designed to enable selfish agents to cooperate on tasks without the need to previously assign a special relationship between them. While the Contract Net protocol is used in many closed systems no common agreed standard emerged for the description of tasks. Furthermore, the approach only allows a binary decision

weather to allocate a task or not; and thus, cannot be used to express a fain-grained and preference-based market scenario.

In more recent years DR-Negotiate and SweetDeal have been developed (Skylogiannisa, 2007; Grosof, 2003). Both are XML based approaches using rules to specify contracts as well as the behaviour of agents in general. The rules can be expressed in RuleML and SWRL, respectively. Both languages have already some kind of standardization, but have not reached the status of a W3C recommendation (Horrocks, 2004). However, DR-Negotiate as well as SweetDeal do not provide any direct approach to model utility maximizing behaviour of agents, but only an institutional corridor to comply with. Since it is possible to express Utility Functions via the introduction of a preferential ranking between different institutional corridors (as it will be discussed in paragraph 3.4) an extension of these approaches is in theory possible, but for a larger choice-set hard to realize.

A further relevant stream of research has been developed from the WS-Agreement Partner Selection approach (Oldham, 2006). The work defines a language as well as a protocol for establishing agreements between two parties. The key feature of this work is that ontologies (vocabularies) are used to facilitate the matching process between offered services and searches for services carried out. Those ontologies allow taking service offers into account that would have been dismissed otherwise. However, the WS-Agreement Partner Selection results only in a set of satisfactory offers of services and not in a ranking between them as needed for a market approach. Thus, this work suffers the same limitations as DR-Negotiate and SweetDeal. Agarwala (2008) and Lamparter have done closely related work in the field of Web Service partner selection. Their approach draws from utility theory in order to evaluate offered services and goods. Utility Function policies are used in the terms of the vocabulary that will be introduced in the next section. The SWRL is used to express rules for the evaluation of offers; goods are described in terms of the Resource Description Framework conforming to the DOLCE upper level ontology (Mika, 2004). Conceptional this is probably closest the approach the one used in this thesis for the evaluation of offers. While the basic idea of using an upper level ontology is to have a proper tool to connect

manifold vocabularies, it is only as good as the level of acceptance of the upper level ontology. However, at least four further approaches are competing; namely, Suggested Upper Merged Ontology, Basic Formal Ontology, General Formal Ontology and WordNet (Niles, 2001; Simon, 2004; Heller, 2004; Fellbaum, 1898). A further back draw of this work is that it demands already at the time of the description of offers to specify which attributes of a service can later be evaluated and which are only part of its context.

Another neighbouring stream of research, Market-Based Multirobot Coordination, originated form the work of Stentz and Dias (Dias, 2000). The driving concept is that a group of robots is given a task, which can be decomposed into subtasks achievable by individuals. A Utility Function defines for each robot the preferences for its resource usage as well as the goals it wants to achieve. Furthermore, a mapping is defined between a Utility Function for the whole team and the individual Utility Function of its members. The interested reader will find a good overview at the field of Multirobot Coordination in Dias (2005). However, coordinating a Multirobot system can differ a lot form coordinating a Multi-Agent System. First, the tasks assigned to robots vary usually from the tasks assigned to software agents. Second, robots often deal with more restricted resources making it easier to apply the market mechanism. And even more, the kind of data produced by sensors of robots is of a different kind than the one software agent's deal with (Zlot, 2004 p.11). Thus, many characteristics differ between mainly non-physical agents and robots, making it difficult to transfer findings from this domain.

The more philosophical research stream of Holonic Multi Agent Systems is inspired by Arthur Koestler (1967). The term "holon" is based on the Greek word "holos" for "whole" and its ending "-on" refers to "part". According to Koestler a Holon is considered as a self-similar structure, i.e. consists of several Holons and is itself part of a greater whole. Heavily stressed analogues are such as a human being consisting of organs, which in turn consist of cells, and is part of a family and wider social structures. The concept of Holonic system design is proposed to integrate Weber's and Taylor's hierarchical top-down structures with decentralized bottom-up

approaches (Warnecke, 1995). The work done by Vázquez-Salceda and Dignum (2005), known as OMNI, is one example for a Holonic Multi-agent System. They show in theory how complex tasks, such as organizing conferences or organ transplantation, can be decomposed into simple tasks, which can be carried out by software agents. However, only very few work has been done to implement such systems; and it remains a vision, yet far from being fulfilled (Gaertner, 2007).

3 Agents and their Policies

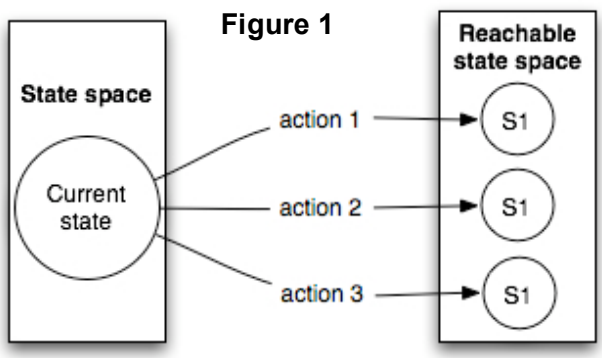
In this section, policies are introduced as a kind of behavioural specification. In the first paragraph the concept of policies in general is in the focus. In the following three paragraphs Action-, Goal-, and Utility Function policies are reviewed successively. In the last paragraph of this section the application side of Utility Function policies is examined.

3.1 The Concept of Policies

A vast amount of definitions on the concept of policy has been published in recent years (Bearden, 2001; Dulay, 2001; Kagal, 2003). Policies play an important role in Multi-Agent Systems since they are a kind of behavioural guidance to determine the decision process and actions of agents. According to Russel and Norwig (1995, p. 37-45) agents can have different levels of behavioural specification. At the lowest level the capabilities as well as the possible range of interactions are limited and hard coded. At higher levels the agents pursue more flexible goals, specified in terms of policies. Kephart and Walsh (2004) presented a unified framework for policies. They distinguish three types of policies covering the range from the lowest level to the highest level of behavioural specification: Action policies, Goal policies and Utility Function policies. Kephart and Walsh's work is based on states and actions, what is quite common in Artificial Intelligence literature and the centre of analysis in this thesis.

In general, a component of a system (e.g. an agent) can be characterized by being in

Figure 1



a state N at each moment in time. Typically, the state N is described as an n-dimensional vector of attributes. Each attribute originates either directly from a sensor or is synthesized from (multiple) inputs (Norwig, 1995, p.36). The application of a policy will directly or indirectly cause an agent to make a transition into a new state. Figure 1 exemplifies this by a scenario in which an agent has the alternative between three actions in the current state N, each of which leads to a possible new state. For instance, the state of an agent, that controls a printer, can be characterized as N={number of queued jobs, available paper, …} and the available actions M as M={accept further job, …}.

3.2 Policies for Reflex Agents

Action policies form the basis of simple Reflex Agents, along with Agents That Keep Track of the History, as described by Norwig (1995, p.38). They dictate the action that has to be taken whenever the knowledge base of an agent is in a given state. Typically Action policies take the form of a rule, where the condition specifies either a fact or a set of disjoint facts that trigger the rule. The consequence of the policy is either the adding or removal of facts from the KB. Since only the "delta" is defined, the state of the KB that will be reached after the triggering is not defined explicitly. However, the designer of the policy is assumed to know which state, using the action, will be entered. Furthermore, the designer of the policy is assumed to consider this state to be more desirable in comparison to states reached upon other rules. While this is necessary to ensure that the agent is behaving as rational as it was designed, it can be argued that the application of Action policies goes along with an unnecessary over-determination (White, 2004). Numerous works on Action policies for the design of agents have been put forward in recent years (Lymberopoulos, 2002; Badr, 2004; Kephart, 2004, to cite just a few). Nevertheless, higher levels of behavioural specifications are needed to enable the mechanisms required for Market-based coordination.

3.3 Policies for Goal-based Agents

Goal policies form the basis of Goal-based agents as described by Norwig (1995 p.40; Kephart, 2004). While Action policies specify exactly what to do in a current state of the KB, Goal policies specify either a single fact or multiple facts that characterize a range of wanted states. Thus, policies of this type

divide the reachable state space in a desirable and in an undesirable one. In other words, Goal policies are not fine-grained enough to express different levels of preference, and every desired state is considered as equally preferable. In the context of Goal policies the agent is responsible to compute actions (generally by the use of IF – THEN statements) that cause the KB to change from the current state into a desired one. While Action policies rely on the designer to explicitly determine what he considered as rational behaviour the Goal policy based agent achieves rationality by perusing a specified goal. This frees from dealing with low-level functions and permits the designer to have greater expressivity. Works put forward on Goal policies include such as (Chandra, 2003; Grosof, 2003; Kephart, 2004; Skylogiannisa, 2007).

3.4 Policies for Utility Maximizing Agents

Utility Function policies codify the agenthood of Utility Maximizing agents (Norwig, 1995 p. 42). They are expressed as functions that value states of the KB. While Goal policies perform a binary classification into wanted or not wanted states they determine a real-valued scalar, expressing the preference for facts (representing the state of the world) in the KB of an agent. Since a Utility Function can be modelled by specifying a set of disjoint goals and specifying a preference relation between them, Utility Function policies can be considered as a generalization of Goal policies from a conceptional point of view (Kephart, 2004). However, such an approach is hardly feasible when there is a large or even continuous state space. In these cases a more compact expression of Utility Functions is needed. Analogue to Goal policies the best state is not predefined at the time of the design of an agent, but is determined during the runtime by selecting the most preferred but still achievable state in a given situation. Goal policies do not allow a fine-grained preference ordering between states making conflicts between multiple Goal policies hard to resolve. On the other hand, Utility Function policies enable the exact specification of tradeoffs; thus, the decision-making in potential conflict situations. However, the challenge in the design of Utility Function policies is to specify an n-dimensional set of facts on which preferences are imposed as well as to find an optimal solution for the Utility Function. Work on Utility

Function policies includes such as (Keeney, 1993; Faratin, 1998; Thomas, 2000; Ermolayev, 2004; Agarwala, 2008).

3.5 The Benefit of Utility Functions

As pointed out Utility Functions are well known in the fields of economics and artificial intelligence as a way of preference specification. The feature of a Utility Function policy, that it maps states to a real-valued scalar will be exploited in this thesis. The designer specifies the Utility Function expressing the values of relevant states. In the next step, given that Utility Function, the agent automatically chooses actions on the designer's behalf.

Consider for example a consumer's preference for a journey has been expressed via a multi attribute function that covers preferred destinations, length of stay, type of hotels and pricing. Different journeys occupy different points in the space of attributes; typically the consumer selects the one with the maximal utility, by him / herself. However, an automated agent that receives a formalized representation of the consumer's Utility Function could select the preferred journey on behalf of the consumer, too. A related scenario is examined by Zou (2003) and Ermolayev (2004). However, Zou uses Goal policies that allow only determining whether an offer satisfies the given conditions or not and lacking the information which of the offers is the best one.

Given that most systems are dynamic in nature, due to changes in the environment and other factors, the feasible actions with the highest associated utilities are likely to shift during the runtime. Thus, agents with Utility Function policies can perform optimization just in time enabling not only more precise decision-making, but also a new type of applications (White, 2004; Das, 2007).

4 A Utility and Trading Model

In this section the basic theoretic concepts needed to fulfil the vision of Market-based coordination are explored in more detail. In the first step it is investigated how an n-dimensional state space can be mapped to a one-dimensional utility value. In the next step it is discussed how to facilitate

sufficient expressivity and domain-independency for the mapping function on the one hand side; while on the other hand side keeping the computational effort at an acceptable amount. In the third step possible interpretations of the concept of utility are surveyed and the reader is taken through an informed decision process on which concept serves best for the end of this thesis. In the last step the focus shifts from the states and their valuation to actions that allow the transition from on state into another.

4.1 States, Attributes and Functions

As pointed out Utility Function policies allow the agent to evaluate possible states of the KB. However, to keep the computational effort at a affordable level not the complete state of the KB shall be evaluated, but only relevant attributes characterizing the state of the KB. Which attributes are considered to be relevant is determined by the policy owned by the agent. This gives the designer the freedom to evaluate new statements from different perspectives, depending on the needs of the agents.

Subsequently it will be referred to a set of statements that describe a potential state of the KB as context C of the attributes A. Thus, a set of attributes $A = \{A_1, A_2, ..., A_n\}$ is a subset of a context C. The technical details of the mechanism that determines which parts of the context C are considered as attributes is examined later in chapter 9. The values a_j of attribute A_j can either be discrete $a_j \in \{a_{j1}, a_{j2}, ..., a_{jm}\}$ or continuous $a_j \in \{\min_j \leq a_j \leq \max_j\}$, as suggested by Faratin (1998). While discrete attributes allow the valuation of properties such as the ability to print in colour or not; continuous attributes can be used, for instance, to express a preference for a low number of queued jobs. The possible attribute space S is an n-dimensional set characterised by the Cartesian product $S = A_1 \times A_2 \times ... A_n$ (Agarwala, 2008). Consequently, Utility Function policies that map the attribute space, within a context, to a real valued scalar are multiple parameter functions (Keeney, 1993 p.219). A concrete set of attributes is denoted by $s \in S$.

A preference structure is usually constructed by a transitive, reflexive and complete binary relation $\succeq$ (Keeney, 1993 p. 141). Transitivity refers to the notion that if a state x is preferred to a state y and a state y is preferred to a

state z then state x it preferred to the state z, too. This can be formalized, in regard to the concepts used in this thesis, as $\forall s_x, s_y, s_z \in S : (f(s_x) \geq f(s_y)) \wedge (f(s_y) \geq f(s_z)) \Rightarrow f(s_x) \geq f(s_z)$. Reflexivity refers to the notion that every set of attributes s is at least as much valued as itself $\forall s \in S : f(s) \geq f(s)$. Completeness states that for every pair of attribute sets $\forall s_x, s_y \in S$ it is either the case that $f(s_x) \geq f(s_y)$ or $f(s_y) \geq f(s_x)$. The structure of preferences can be inferred from an evaluation function $f : S \rightarrow R$, that maps a state to a real-valued scalar, where the condition $\forall s_x, s_y \in S : s_x \geq s_y \equiv f(s_x) \geq f(s_y)$ holds.

4.2 Additive Utility Functions

In the previous paragraph a most general form of the function $f(S)$ that maps each possible combination of attributes to a real valued scalar has been introduced. However, the number of possible attribute combinations grows exponentially with respect to the attributes and their values (Keeney, 1993 p. 283). Consider for example that there are 5 attributes with 6 discrete possible values; then the size of the possible attribute space S is 5^6=15625. While such an approach is theoretically possible it is practically not feasible. Fortunately, in many practical situations the attributes are mutual independent, also known as additive severability (Chevaleyre, 2004). The assumption of additive severability does hold if and only if there exist functions $f_1, f_2, ..., f_n$ such that $f(a_1, a_2, ..a_n) = f_1(a_1) + f_2(a_2) + ... + f_n(a_n)$. Relying on additive severability improves the compactness and manipulability by decomposing $f(S)$ into multiple one-dimensional functions. The concept of additive Utility Functions is widely used in the field of Multi-Agent Systems and is, in depth, explored by Faratin (1998) and Chevaleyre (2004).

4.3 Cardinal versus Ordinal Utility Theory

After lining out the theoretical details of Utility Functions, the nature of the utility value itself has to be closer examined. In economics two different notions of the utility concept can be distinguished; the ordinal utility concept and the cardinal utility concept (Keeney, 1993 p. 26). The ordinal utility concept captures only the ranking and not the strengths of preferences. On the other hand in cardinal utility theory the magnitude of utility differences is

treated as a significant quantity. When an ordinal utility notion is used differences in the valuations carry only the information of the preference ordering between the members of a choice set (i.e. the different sets of attributes whose utilities should be compared). Thus, ordinal utility theory can only be used to determine which of the alternatives is the best one (Oberender, 2004 pp. 79).

However, in many cases the information which feasible state is the best is not adequate but the knowledge weather even the best option is sufficient is of use, too. The cardinal utility concept carries the objective utility of a concrete set of attributes; thus, allows to decide whether a set of attributes is describing something whose utility is big enough. In particular, the cardinal utility concept allows decision making without the need to construct further options to compare with that are necessary if an ordinal utility concept is used. Even more important however, the cardinal utility concept allows the comparison of levels of satisfaction across different agents; if a particular item gives one agent 4 utility units but another gets 2 from the same thing, the item described by a set of attributes is said to give the one twice as much as the other one (Oberender, 2004 pp. 109).

This sort of comparison is of great use in Multi-Agent Systems when it comes to the evaluation of how well the system is doing overall. Under the framework of utilitarianism actions are judged by their contribution to the reaching of the overall goal of the system. The concept of cardinal utility provides a way to search for what has been called by the Philosophical Radicals the “greatest good for the greatest number” (Hardin, 1968); thus, to search for a state in which all goals of all agents are best satisfied. This ability to compare utilities between individuals runs into ethical problems in the case of human agents. However, it shall be assumed that these ethical problems are not of importance in the case of software agents. For a further investigation of different notions of utility, in particular on the widely used disjunction between the total and average utilitarianism, see Sen and Williams (1982).

4.4 The Auctioning Process

In the previous paragraphs a mechanism that is potentially capable of determining the value of actions in terms of utility, associated with the state reached upon the application of an action, has been introduced. However, a negotiation mechanism is needed if the demand for something is beyond the limitations of a finite set of available resources. A negotiation mechanism can be seen as a protocol prescribing how agents interact to determine a contract granting one agent access to a resource and excluding another one. Auctions are a type of those protocols as characterized by McAfee and McMillan (1987): “An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market principals.” The most common encountered types of auctions are the English auction, the Dutch auction, the First-price Sealed-bid auction and the Second Price auction. A general introduction to auction theory and surveys of several different types of auctions is provided by Wolfstetter (1996).

This thesis uses the mechanism of the Second Price auction, also known as Vickrey auction, in which bidding agents submit their bids without knowing the bids of the other agents. The highest bidding agent wins, but the price to be paid is the one in the second highest bid. The Vickrey auction belongs to the category of the Vickrey Clarke-Groves Mechanism (Clarke, 1971; Groves, 1973). The idea of this mechanism is that each agent in the auction pays the opportunity costs that their presence puts on all the other agents. Payments between agents are done by the use of Util’s. Whereby, the valuation function $f(util) = util$ neglects the diminishing marginal utility of an additional Util for the sake of simplicity and to use the valuation derived by a Utility Function as the willingness to pay. Consider for example, agent A auctions one printing job, and two agents are interested in printing. Agent B wants to print and is willing to pay 10 Util’s for it. Agent C wants to print and bids 6 Util’s to get that job done. In the first step, the outcome of the auction is decided in favour of the highest bidder: the printing opportunity is allocated to agent B. In the next step the opportunity costs are considered that the winning agent imposes on the other bidder(s) to decide the payment. Currently, agent C has a utility of 0

and agent B has a utility of 10. If agent B had not been bidding, agent C would have won and had a utility of 6; thus, B pays 6 Util's.

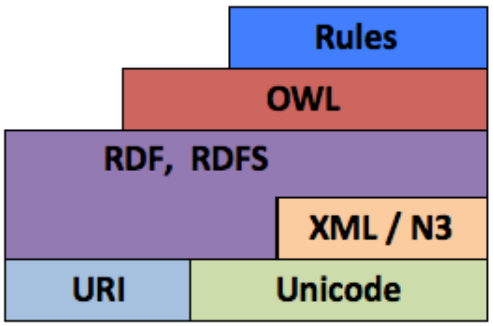
Vickrey auctions have the properties of self-revelation / incentive compatibility and ex-post efficiency (MacKie-Mason, 1995). Self-revelation is given if no incentive exists to collect information about competitors and to lie is present. Even more, in the case of Vickery auctions each bidder maximizes its utility upon bidding (revealing) its true valuation, freeing this thesis from the need of further Game-theoretic enquiries on optimal bidding strategies. Ex-post efficiency means that the winning agent is the one with the highest valuation.

5 The Semantic Web

In this section the medium in which the agents operate in is investigated. While there are a vast amount of "hands-on" publications about the semantic web (Burners-Lee, 2001; Hendler, 2001), the approach followed here will be a more theory oriented one. In the first step the vision of the web of data is shortly outlined. In the following paragraphs the building blocks of the Semantic Web are introduced layer after layer. In the last paragraph the reader is made familiar with the distinctions between domain and application ontologies used in this thesis.

5.1 The Vision

Figure 2



Originally the Web was envisioned as a set of tools for the representation of relationships between named objects (Berners-Lee, 1999, p.41). Now, the main aim of the Semantic Web initiative is to fulfil this vision. The collaborative efforts aim at the development of technologies and standards, which help machines to understand the meaning of data. The guiding idea is having data linked in such a way that it can be retrieved and reused across platforms, applications and communities (Berners-Lee, 1998, p.9; Koivunen, 2001). To meet these

goals, in a collaborative effort led by the World Wide Web Consortium, a set of layers has been designed (figure 2).

5.2 Unicode and URI

At the bottom, Unicode and URI follow the features of the existing World Wide Web. Unicode is a standard allowing computers to consistently represent and manipulate text. A Universal Resource Identifier is a string of characters to identify or name a resource. Whereby a resource can be everything; from an object, only existing in the WWW, to a real human person (Ding, 2005). The usage of URI is important for distributed systems like the WWW, since it provides integration of all resources (Dürst, 2007).

5.3 The XML layer

At the core level is the XML layer with its XML Namespace and XML Schema, providing the syntax for the Semantic Web. The primary purpose of XML is to facilitate the encoding and serialization of structured data. XML is classified as an extensible language as it allows its users to define their own elements (Bray, 2006; Dürst, 2007). XML namespaces allow users to provide exclusively named elements and attributes in an XML document. An XML document can contain element or attribute names from more than one XML vocabulary. If each vocabulary has been given a namespace then the ambiguity between identically named elements or attributes can be resolved. XML schema can be used to express a set of rules to which an XML document must conform in order to be considered valid according to that schema (Fallside, 2004).

5.4 The Metadata Level

On the metadata level RDF and RDFS allow the description of resources. The Resource Description Framework is a specification designed for representing metadata about resources in a graph form (Manola, 2004). However, RDF has become a general method of modelling information through a variety of syntax formats.

The normative syntax for serializing RDF is XML in the RDF/XML form, but other serialization formats, such as Notation3 that will be discussed in section 7 in detail, are used as well. The RDF data model is based on triples in the

form of “subject-predicate-object” expressions. Hayes (2004) defines the formal semantics of RDF. As pointed out Resources can represent everything, from websites to people. All elements of a triple are resources with the exception of the object and subject that can be also literals. Literals can be either plain literals or typed literals using XML Datatypes as defined by Biron (2004). Multiple triples together form a RDF graph. A subject or object without URI is called blank node and can be viewed as a graph scoped identifier that cannot be directly referenced from outside the document. RDF reification allows the dissembling of a triple to its parts and to use these parts as objects in other statements; in other words, the citation of graphs within graphs. Reification is the expression of RDF triples by the use of RDF in such a way that the language becomes treatable by itself (Brickley, 2004). This mechanism allows looking at a RDF graph and reason about it, using RDF tools (Berners-Lee, 2004). Similarly to a XML document, a RDF document can have a schema (RDFS) to conform with.

RDF Schema extends the RDF vocabulary to allow the description of taxonomies. Furthermore, it extends the definitions of some RDF elements (Brickley, 2004). By the use of RDFS, all resources can be typed to groups called classes. Such typed resources are instances of a class. Classes are resources, too; thus, they can be described by properties and identified upon URI’s. The set of instances of a class is called extension of a class; whereby two different classes can share the same instances. The properties in RDFS are relations between subjects and objects in RDF. Properties may have a defined domain to declare the class of a subject in a triple using this property as predicate. Properties may also have a range to declare the class or XML Datatype of the object in a triple using this property as predicate.

5.5 The Ontology level

Both RDF and RDFS provide basic features of knowledge representation, but more modelling primitives are needed. The Web Ontology Language OWL enables the semantic specification and conceptualization of different application domains. The aim of OWL is to bring the expressiveness of description logics to the Semantic Web. The W3C OWL recommendation

includes the definition of three variants of OWL with different levels of expressiveness (Bechhofer, 2004).

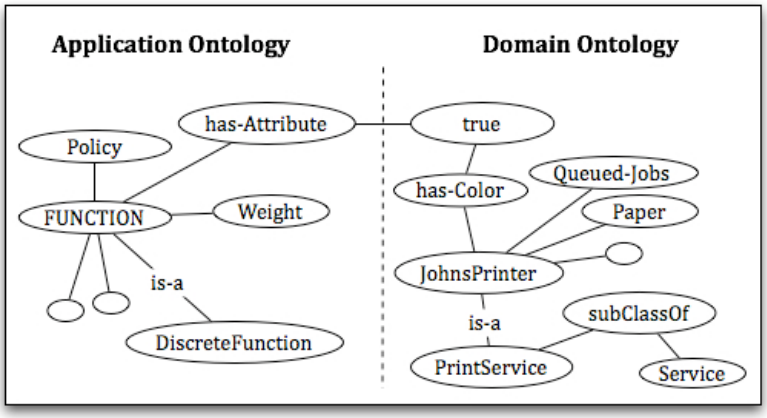
OLW Light provides classification hierarchies and simple constraints, such as 0 and 1 cardinality. Originally it was hoped that it would be simpler to provide tool support, allowing a quick migration path for existing systems (McGuinness, 2004). OWL Light is the simplest OWL language and corresponds to description logic SHIF (Eiter, 2004 p. 82).

OWL DL is intended to support a maximum of expressiveness possible, but still retains computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time). OWL DL corresponds to description logic SHOIN (Eiter, 2004 p. 84). The language variant includes all OWL constructs, but they can be used only under certain restrictions. For example number restrictions are not allowed to be placed upon properties that are declared to be transitive.

OWL Full is designed to utilize the full syntactic freedom of RDF and RDFS. It is funded on semantics different from OWL Light and OWL DL; it has no expressiveness constraints, but does no longer guarantee the properties of computational completeness and decidability (McGuinness, 2004). For example, in OWL Full resources can be treated simultaneously as a class and as an individual.

5.6 Two Ontologies for One Purpose

Figure 3



In the previous paragraphs of this section the technical underpinnings for a formal representation of knowledge as a set of resources and the relationships between those resources have been introduced. Such formalism can be used to reason about the properties of a

domain, and may be used to define a domain. In this paragraph, two, by their use distinct, types of ontologies are investigated; the first one, domain independent application ontologies; the second one, domain ontologies. Generally speaking the application ontologies, used in this thesis, are in the namespace “org”, the domain ontologies, used in the examples, are in the namespace “ex”.

The application ontologies describe how mathematical functions and policies have to be made up so that an inference engine is able to process them. These application ontologies are designed only for one application, but are domain independent in such a way that the application itself is not restricted to one specific domain to deal with.

On the other hand side domain ontologies model a specific domain, or part of the world. Domain ontologies define the particular meanings of the classes and predicates as they are apply to a domain. Thus, domain ontologies are used, in this thesis, to describe auctions, offers, and bids as well as states of the KB of agents.

6 The used Platform

In this section the building blocks of the agents, used in this thesis, and the platform on which they operate on are described. In the first step the idea behind the platform is shortly presented. In the next step the reader is invited to have a look under the hood of the Platform, on FIPA standards and the Jade technology. In the last step the layers of the Ubiware agent are investigated.

The reader is asked to recognize that, while this section is not in the centre of research in this thesis, this section is necessary to understand the concepts built on top of this platform, introduced later in this thesis.

6.1 The Vision of the Ubiware Platform

The driving idea behind the Ubiware Platform is that a software agent can represent every component of a computing system. The realization of such a vision requires a flexible core consisting of autonomous components (Bai, 2006; Terziyan, 2007). Social level characterisation of MAS and ontological

approaches to coordination are two important research directions, which try to deal with the question of the balance between the freedom of the individual agent and the predictability of the whole system. The social level characterisation addresses the need for understanding the impact of the organisational context on the individual agent as well as the emergence of social patterns from individual behaviour (Vázquez-Salceda, 2005 pp. 309). The ontological approaches focus on enabling agents to communicate and reason about actions, plans and knowledge (Tamma, 2005; Boella, 2004). The core of the UBIWARE Platform is based on findings of both research directions and designed to integrate them (Artem, 2007; Terziyan, 2007).

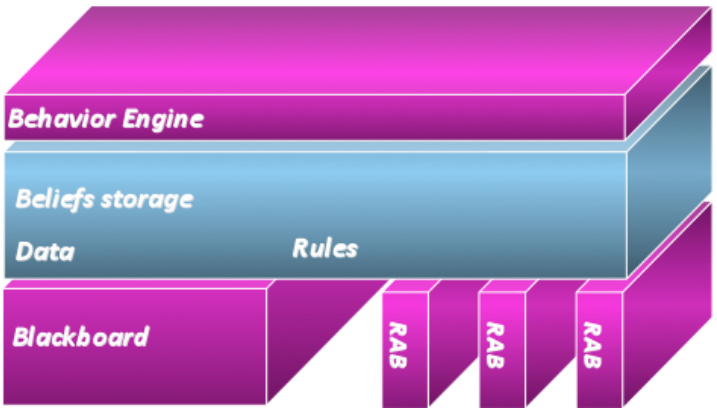
6.2 FIPA and Jade

The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit organization whose purpose is to develop specifications of technologies that ensure interoperability within and across platforms. The FIPA 97 specifications declare normative rules that allow agent societies to operate, communicate and to be managed (Bellifemine, 2001 p. 90). The Agent Management System provides “white-pages” services and life-cycle management. The Directory Facilitator provides “yellow-pages” services to the agents. The Agent Communication Channel ensures the interoperability within and across platforms. However, the FIPA specifications are not intended to be a complete blueprint for building Multi-Agent Systems. For example, the FIPA specifications do not advice how to model existential aspects of the agents (Charlton, 2000).

The JADE Framework is an open source software framework that supports the development of interoperable multi-agent systems in complaint with the FIPA specifications (Bellifemine, 2001 p. 93). Thus, JADE implements the Agent Management System (AMS), the Directory Facilitator (DF) and the Agent Communication Channel (ACC). The agents are realized as Java threads and come with a FIPA-compliant Global Unique Identifier for each individual agent on the platform. The platform can be grouped into containers and distributed on several hosts; furthermore, mobility of the agents between the containers is supported.

6.3 Layers of a UBIWARE Agent

Figure 4



The UBIWARE Platform is build on top of JADE and is the successor of the Smart Resource Platform. The UBIWARE agent is an extension to the JADE agent in such a way that it adds three layers: Reusable Atomic Behaviours, Believe Storage, and a Behavioural Engine, see figure 4 (Terziyan, 2007).

The first layer consists of the Reusable Atomic Behaviours (RAB's), pieces of Java code implementing basic functions. The Reusable Atomic Behaviours incorporate the perceptors and actors of agents as described in (Norwig, 1995, p. 17). Reusable in this context means that these code blocks can be applied across different agents and different scenarios in different applications.

The second layer of an agent is the Believe Storage. The behaviour of an agent is determined by the roles it enacts and believes it has about the world (i.e. KB), both stored in the believe storage. Possible examples of roles are the Platform-starter, Seller or Buyer. While one agent usually plays several roles different agents can play one role. The roles are encoded in the Semantic Agent Programming Language. S-APL uses the RDF data model, i.e. everything is structured as a set of "subject-predicate-object" triples. These triples represent the knowledge of agents and specify the conditions and parameters for the execution of the RAB's necessary for playing a role (see more in the next section).

In the third layer there is the Behaviour Engine. The engine consists of the agent core and the two behaviours "Live" and "Assign Role". The Live behaviour iterates through all behaviour rules, checks them against existing believes and executes the appropriate rules. The enacting of a rule usually includes the execution of a set of actions, such as adding and removing

believes and the carrying out of RAB's. The Assign Role behaviour phrases the S-APL documents and loads them into the believe storage. Furthermore, it registers new roles with the JADE DF agent. The Assign Role behaviour has the duality of being a part of the behaviour engine as well as being a RAB at the same time since, the creation of an agent needs at least one behaviour model and all later invocations of Assign Role are specified in some behaviour models.

The UBIWARE Platform allows the agents to access behaviour models either from a central repository or from a distributed environment, like the World Wide Web. This approach to updating of the behaviour models brings multiple advantages. At the one hand side not pre-programmed roles can be loaded at runtime. At the other hand side the roles can easily be changed and updated. Corresponding to the behaviour models the platform enables its agents to access RAB's during runtime. Thus, if the enacting of a role prescribes to execute a RAB the agent is missing, the RAB can be downloaded from a repository (Artem, 2008).

7 The Semantic Agent Programming Language

In this section the language used by the agents is introduced. While the first and the second paragraph follow the classic distinction between Knowledge and Rule Base in Knowledge Based Systems, the reader will recognize that such rigid distinctions are not necessary, in the case of the Ubiware platform. Since the agents are not isolated but only one part of a large computing environment, the next paragraph gives some insights on question of the integration of the agents. In the last paragraph build in functions are introduced, that will be widely used in the following sections.

Likewise to the previous section this section is not in the research focus of the author. However, to understand the subsequent steps, made in this thesis, this section is crucial. Since the following paragraphs are mainly based on Katasonov (2008), quotations are only made when other sources are used.

7.1 The Knowledge Base

The Semantic Agent Programming Language is the language used for the Production System (i.e. Behavioural Engine) in the UBIWARE agents. S–APL is based on the syntax of Notation3. N3 allows RDF to be expressed, but is compared to the dominant RDF/XML syntax more compact and better human readable. Notation3 is not an official W3C Recommendation, yet; however, a complete specification is given in the Design Issues of the W3C (Burners-Lee, 2006). In line with the RDF data model the atomic sentences in S-APL are *rdf:Statement*'s. Please note that other authors refer to *rdf:Statement*'s as believes (Katasonov, 2008). However, besides the different naming, statements as well as believes refer to the same concept.

Each statement has a subject-, predicate- and an object part. Due to this structure statements are also called "triples". Another useful way to think about RDF is as a graph, in which objects and subjects are nodes, connected by a predicate as edge. An example for an RDF statement is:

<http://example.com/socrates>
<http://www.w3.org/2000/01/rdf-schema#type>
<http://example.com/Human>.

URI terms can be abbreviated using namespaces; the empty namespace is used as default namespace. However, S-APL is not restricted to the use of URI's but allows literal terms in the subject as well as in the object part of the triples.

@prefix ex: <http://www.example.com#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
ex:socrates ex:hasAge "32"^^xsd:int.

While in the case of standard RDF every statement is treated as a global truth; S-APL allows the quotation of statements. In addition to the concept of reification in RDF, the quotation of statements is done by the use of containers, either in the subject- or in the object part of a statement. Syntactically, a container begins with an opening bracket "{" and ends with an closing bracket "}". Consider for example:

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

{ex:socrates rdfs:type ex:Human} :accordingTo ex:tim.

The quotation of formulas by the use of containers restricts the validity of a statement to its container. Thus, statements have any meaning only inside their containers. The most general root container, which is not cited by any other statement, is referred to as “G”. The citation of containers allows the construction hierarchies, with the general container G as their root. Whereby, every container has to be linked to the root container through a hierarchical chain to give meaning to it in respect to G. In the graph model of RDF a container is a node that holds instead of an URI or a literal a complete graph on its own. As shown in the example above, a container allows the distinction between what an agent believes to be true and what someone else, including other agents, does believe.

7.2 The Rules

The most important aspect about the concept of quotation and container is however, that their use allows to add statements to the root G under certain conditions; thus, the constructions of Production Rules in the form of IF - THEN statements. More precisely, is the content of a container found in the root Container G and is the container connected, as a subject, with another container via an implication predicate, then the content of the object container is added to the root container of the KB as well. For the purpose of implications S-APL uses the concept of variables as applied in N3Logic (Berners-Lee, 2008). Variables are placeholders that are bound in the THEN part (and all its nested sub-containers) of a Production Rule, according to the matches in the IF part. The central idea is that, given the variables, a Production Rule is a relationship between two graphs within containers. Production Rules are expressed in S-APL using the predicate “=>” as a short for *<http://www.ubiware.jyu.fi/sapl#implies>.* They are inferred only if the implication predicate => is part of the root container G. As in the programming praxis there are frequently occurring exceptions it should be noticed that implications are although inferred if they are in a container that is connected to the general context by the statements *{…} sapl:is sapl:Rule or {…} sapl:requires* *. The IF - part in the subject of the Production Rule is called antecedent graph, and the THEN - part in the object is called consequent

graph. The domain as well as the range of the implication predicate "=>" is sapl:Container. In the simplest case the implication is a one shot rule, and together with its antecedent graph removed from the believe storage after the rule has fired, while the consequent graph is added to the root container. The following example illustrates this in greater detail:

@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.

{?x :accordingTo ex:tim} => {?x sapl:is sapl:true} .

If a statement matches the IF - part of the rule, such as the one above, the container bound to the variable ?x is added to the root container G upon the statement in the THEN part *?x sapl:is sapl:true*. In many situations however, not only one match for a variable is possible. To allow the implication to be inferred for all possible matches in the IF - part of a Production Rule the IF - part has to be wrapped into the container *{} sapl:All ?x*. Whereby ?x is the variable for which all possible matches shall be considered. However, the *sapl:All ?x* construct is still not a persistent rule; but does only guarantee that all matches in the IF - part will also be treated in the THEN part; after such a rule has fired it gets removed in the same way as the rule in the previous example. To build a persistent rule the implication has also to be wrapped into the container *{…} sapl:is sapl:Rule*. Consider the following example.

{{{ ?x rdfs:type ex:Human. } sapl:All x?.

}=>{

?x rdf:type ex:Mortal.

} } sapl:is sapl:Rule.

Special kinds of rules are the entailment rules necessary for the RDFS / OWL vocabulary which can be expressed analogue to the previous example. The following formula expresses the semantics of the *rdfs:subClassOf* property as determined by the rdfs9 entailment rule (Hayes 2004). The construct *sapl:I sapl:doNotBelieve {…}* in the antecedent graph is used to ensure that the implication is only inferred if the consequent is not part of the agents KB; thus to avoid unnecessary computations.

{{ ?S1 rdfs:subClassOf ?O1.

?S rdf:type ?S1.

```
        sapl:I sapl:doNotBelieve {?S rdf:type ?O1}.
} =>{
        ?S rdf:type ?O1.
}} sapl:is sapl:Rule.
```

In combination with the statements describing the relationships between the concept – called ontology - of being human and being mortal

```
ex:Human rdfs:subClassOf ex:Mortal.
```

it is possible to infer further statements about instances that are members of the class ex:Human (Manola, 2004), such as:

```
ex:socrates rdfs:type ex:Mortal.
```

Reminding that quoting allows one to express relationships between RDF graphs within different containers, for example stating that a given RDF graph has the provenience of a particular document. Every RDF graph is composed of multiple statements. A quoted statement is enclosed in brackets "{","}", representing a container.

7.3 Integration with existing infrastructure

The World Wide Web is designed as a mapping between URIs and the information gained when such URIs are resolved by the use of appropriate protocols (Rosenfeld, 2002 p. 56). In S-APL a information resource is identified by a symbol, which is either pointing to a local repository or a URI. *:missingModel* is a property that causes a RDF graph corresponding to the resource provided as object to be added to the container given as the subject. In order to access a S-APL script, named hello.sapl, from the local repository and add it to the root container of the agent's KB *:missingModel* is used in the following manner:

```
sapl:I :missingModel hello.
```

Please note that the resource sapl:I is a synonym for the agent ID. The following example accesses a S-APL script by resolving its URI and stores it's content to the container with the ID "C":

```
C :missingModel <http://www.example.com/hello.sapl>.
```

The advantage of loading RDF graphs to a special container is that it allows rules to access the Web, and to check the content of RDF documents, without loading them into the general context and believe everything stated in them.

Since the predicate *:missingModel* has no predefined meaning its functionality has to be determined first. The first step is, as previously pointed out, the construction of a Production Rule, with *?container :missingModel ?model* in the antecedent graph. In the consequent graph the invocation of a Reusable Atomic Behaviour (RAB) provides the desired functionality of loading the script to a container. Note that analogue to implications RAB's are only scheduled for execution if they are part of the root container of the agents believes storage. After the RAB has been scheduled for execution its representation, in the root of the KB, gets removed. The following example depicts the invocation of the BeliefsLoadBehavior with a simple configuration.

```
@prefix java: <http://www.ubiware.jyu.fi/rab#>.
@prefix p: <http://www.ubiware.jyu.fi/rab_parameters#>.
{
        ?container :missingModel ?model
}=>{
        {sapl:I sapl:do java:ubiware.shared.BeliefsLoadBehavior}
        sapl:configuredAs {
                p:saveTo sapl:is "?container"^^xsd:string.
                p:inputFromFile sapl:is "?model.sapl".
}}
```

While the "*java:*" namespace is used to determine that the object is a RAB; the following string indicates the class path, i.e. where the java code can be found. The "*p:*" namespace in the configuration part of the RAB defines parameters, such as the container to whom the statements should be added or the location of the file that contains the relevant data.

7.4 Build in functions

While N3 properties can be used simply as ground facts it is also necessary to have the possibility that they can be calculated, too. Literals can be evaluated against the agent's engine and assigned to a variable. These built-in functions

can be used to provide a variety of functionality such as string matching and mathematical functions as used in the next section. The following triple calculates the length of the string bound to the variable provided to the function in the object and binds the result to the variable in the subject part of the triple:

?y sapl:expression "length(?x)".

8 Modelling Mathematic Functions

In this section the functions for the evaluation of attributes are introduced. The application ontology used for the functions is widely inspired by the work done by Agarwala (2008) in terms of the DOLCE ontology. To acknowledge the different types of attributes (i.e. discrete and continuous ones) not only one function is introduced, but three. In the first paragraph Discrete functions are introduced. In the next paragraph, Linear functions are introduced. In the last paragraph Polynomial functions are introduced. To give the reader a broad overview on the design of the functions figure 5 shows their ontology in graph form.

Figure 5

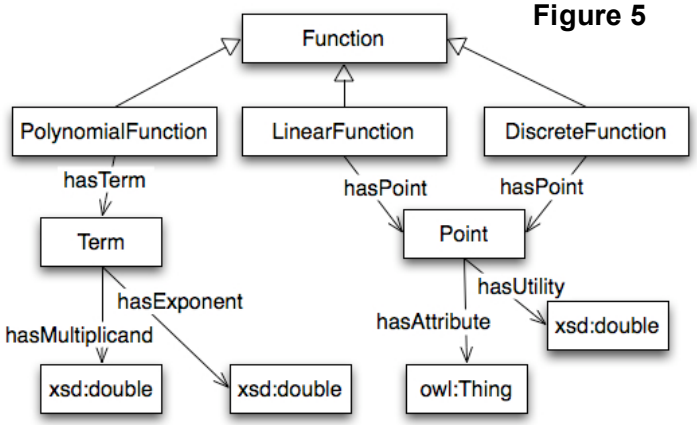
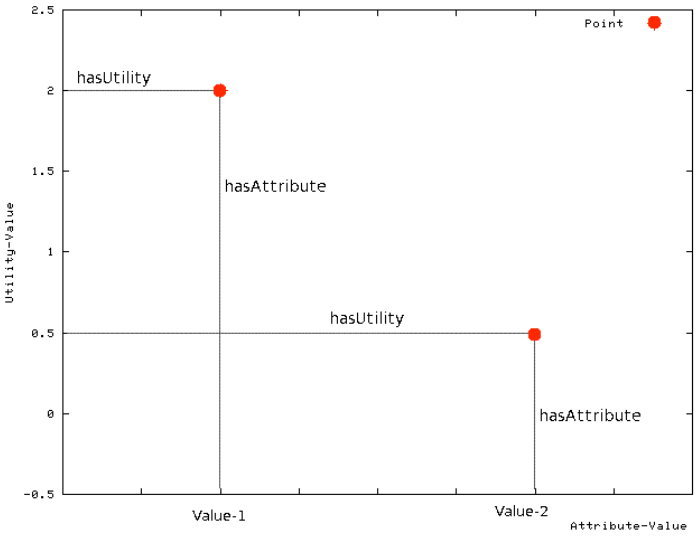


Figure 6



8.1 Discrete Functions

Functions of the *rdfs:type org:DiscreteFunction* are connected to a set of individuals of the rdfs:type org:Point, by the predicate org:hasPoint. Each individual point is connected via the property

org:hasAttribute to a *org:Attribute,* and via the datatype property *org:hasUtility* to one utility value of the type *xsd:double*. Subclasses of *org:Attribute* can be used to define which values a specific attribute can adopt. Axiom A.1 and A.2 formalize this notion.

$$org:Point \subseteq owl:Thing \sqcap_{=1} org:hasAttribute.owl:Thing \sqcap_{=1} org:hasUtility.xsd:double \quad \text{(A.1)}$$

$$org:DiscreteFunction \subseteq org:Function \sqcap_{>0} org:hasPoint.org:Point \quad \text{(A.2)}$$

In the next step rules are required to evaluate the introduced concepts and their semantics. The formula below checks the set of points, of a discrete function f, to find the utility value for a given attribute value; thus, it declares how the value of a attribute can be determined.

```
{       sapl:I :calculate {?a ?f *}.
        ?f  rdf:type org:DiscreteFunction;
                org:hasPoint ?p.
        ?p org:hasAttribute ?ar;
                org:hasUtility ?u.
        ?a = ?ar.
} => {
        sapl:I :calculate {?a ?f ?u}.
}
```

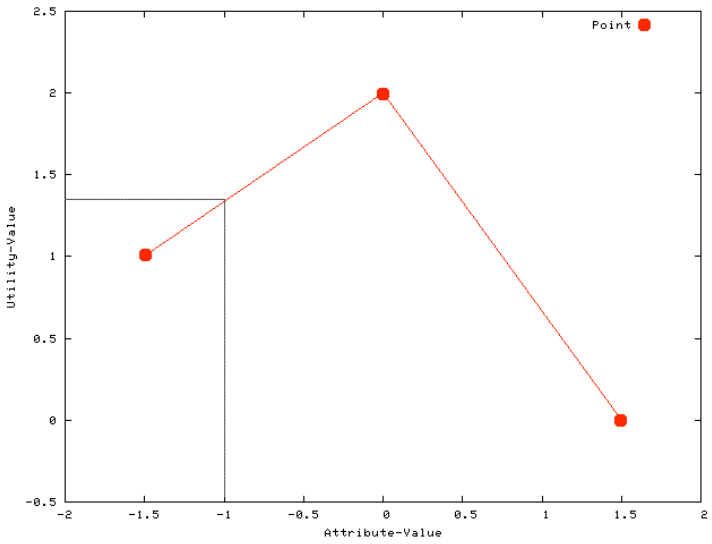
In the formula above the testing whether a given fact satisfies the attribute defined by a point is done by a simple equals-relationship. However, different types of attributes require different rules of entailment. Thus, this predicate should be altered according to the type attribute that is evaluated. While in some cases simple string matching is sufficient, other attributes demand a test whether at least the *rdfs:subclass*-relationship for the type of the attribute holds. The complete formula used in the prototype can be found in Appendix A.1. A in depth discussion, about further possible entailments, for concepts is given by Bernstein and Kiefer (2006).

Ontologically modelling Functions is a relation between an attribute, an instance of the class org:Function, and a valuation to which the later maps the

former. To represent such a three value relation the container in the statement sapl:I :calculate _:containerID has in its subject part the attribute, in its predicate part the Function URI, and in its object part the valuation of the subject by the predicate. The construction of this container is done in the THEN-part of the rule by the statement *sapl:I :calculate {?a ?f ?u}.*

8.2 Linear Functions

Figure 7



Linear functions allow an ordering between given attribute values, of the type xsd:double, to specify a continuous range. They are an extension of the already introduced discrete functions in that way that points with neighbouring value attributes are connected by a line to make up a linear function. For each pair of points (x_1,v_1) and (x_2,v_2) as well as a given attribute value x, a utility value is calculated by the equation: $u = [(v_2 - v_1) * (x - x_1)]/(x_2 - x_1)$, for $x_1 < x \leq x_2$. Axiom A.3 formalizes the ontological status of the class org:LinearFunction.

$$org:LinearFunction \subseteq org:Function \wedge_{>1} org:hasPoint.org:Point \qquad \text{(A.3)}$$

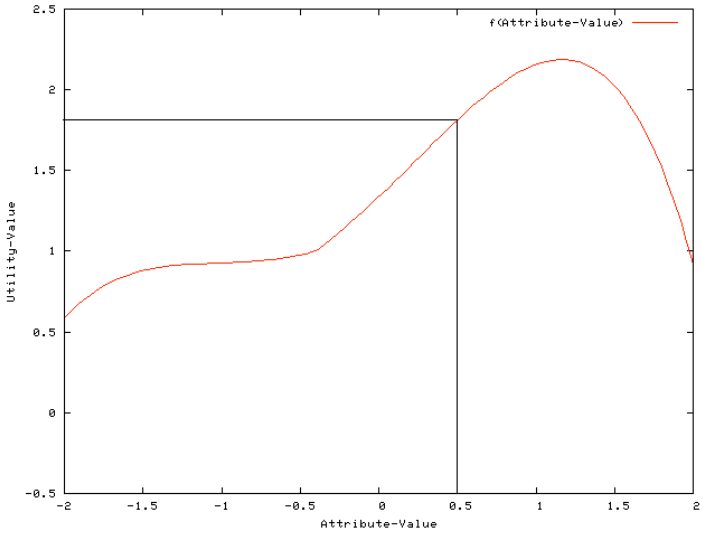
To evaluate attributes with linear functions the build in math predicates are exploited.

```
{       sapl:I :calculate {?a ?f *}.
        ?f rdf:type org:LinearFunction;
              org:hasPoint ?1p;
```

```
        org:hasPoint ?2p.
    ?1p org:hasAttribute ?1v;
        org:hasUtility ?1u.
    ?2p org:hasAttribute ?2v;
        org:hasUtility ?2u.
    ?1v < ?a.
    ?2v > ?a.
    ?v1v sapl:max ?1v.
    ?v2v sapl:min ?2v.
    ?1v = ?v1v.
    ?2v = ?v2v.
    ?u sapl:expression "?1u+(?a-?1v)*((?2u-?1u)/(?2v-?1v))".
} => {
    sapl:I :calculate {?a ?f ?u}.
}
```

The formula, given above, calculates the utility u of a given attribute value v and ensures that only neighbouring points are taken into account for the calculation. The adding of the result of the function, to the KB, is analogue to the last paragraph. The complete formula, used in the prototype, is given in Appendix A.2.

Figure 8



8.3 Polynomial Functions

In addition to the previous paragraphs functions for continuous facts, such as the float datatype, can be modelled by means of polynomial functions. The class *org:PolynomialFunction* denotes functions constructed from one or more variables and constants

using the operations of addition, subtraction, multiplication, and raising to constant non-negative integer powers. In general it can be written $f(x) = a_1 x^{p_1} + a_2 x^{p_2} + ... + a_n x^{p_n}$, where a_n and p_n represent parameters that have to be given to create the function.

The ontology of polynomial functions is defined as follows. Instances of these function are connected to a set of terms by the predicate *org:hasTerm*; each individual term has exactly one multiplicand and one exponent. Axiom A.4 in conjunction with A.5 formalizes this notion.

$$org:\text{PolynomialFunction} \subseteq org:\text{Function} \wedge org:hasTerm.org:Term \quad \text{(A.4)}$$

$$org:Term \subseteq owl:Thing \wedge_{=1} org:hasMultiplicand.xsd:double \wedge_{=1} org:hasExponent.xsd:double \quad \text{(A.5)}$$

The following formula calculates the sum of all terms specified for a polynomial function as well as the value of the terms itself. The adding of the result of the function is done likewise to the previous formulas. The complete version, used in the prototype, can be found in Appendix A.3.

```
{{      sapl:I :calculate {?a ?f *}.
        ?f rdf:type org:PolynomialFunction;
                org:hasTerm ?p.
        ?p org:hasMultiplicand ?w;
                org:hasExponent ?e.
        ?x sapl:count ?p
        } sapl:All ?p.
} => {
        {       ?u sapl:expression "?w*pow(?a,?e)"
        }=>{    ?p :hasResult ?u}.
        {       * :hasResult ?zu.
                ?y sapl:count ?zu.
                ?y = ?x.
                ?h sapl:sum ?zu.
        }=>{
```

sapl:I :calculate {?a ?f ?h}

}}.

9 Mapping a Graph to a Value

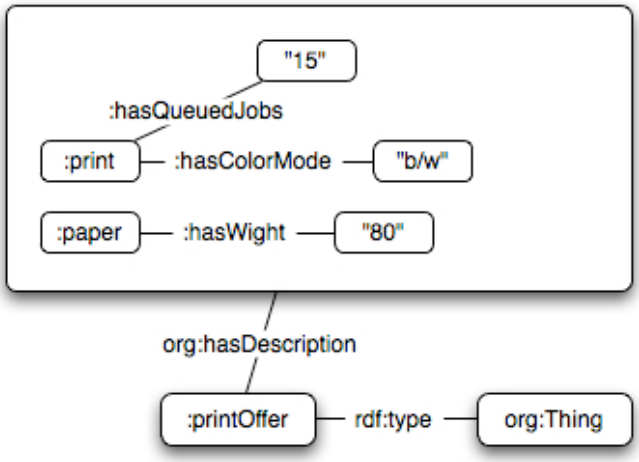
Preferences of an agent towards states of its Knowledge Bases can be expressed upon Utility Functions that map a set of RDF statements, describing the current state of its KB, to a real-valued scalar. These Utility Functions provide a view on the world, from the perspective of the agent in the way that they reduce the complexity of the state of the world, of which the KB of the agent is a model of, to a one-dimensional utility valuation. Such utility valuations can be used to compare two states of the world. Thus, the comparison of utility valuations, for states of the KB before and after an hypothetical action indicate the value of an action for the agent.

In the first paragraph of this section the application ontology for the declaration of the statements describing the states of the world is presented. In the next section the makeup of multi attribute functions, that evaluate the state of the world, is introduced. After that the formula for the evaluation of multi attribute functions is presented. In the last paragraph it will be explored how already existing complex functions can be refined, via the concept of inherence.

9.1 A Makeup for States Of The World

In this paragraph a makeup for states of the world is introduced. Individual states are made up by the class *org:Thing*. Each state has exactly one container associated, describing what the state is, as formalized in by axiom A.6. For example consider a printer controlled by the agent; the description

Figure 9

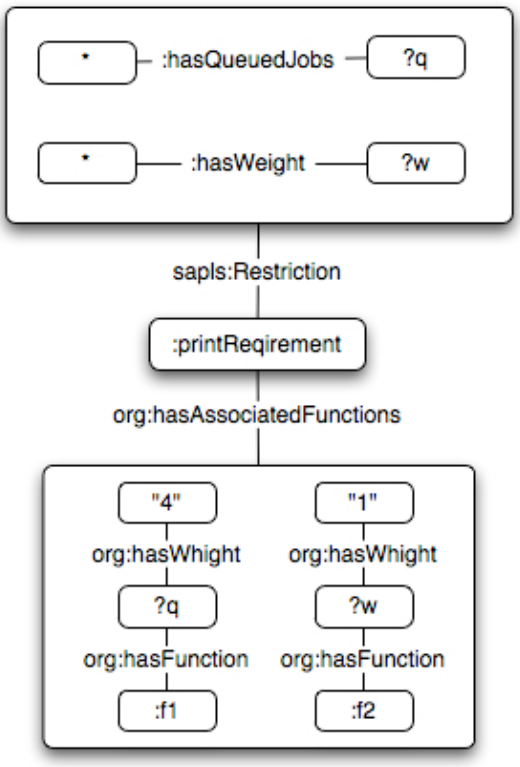


container can host information about the used paper, about the availability of colour or the number of queued print jobs.

$$org:Thing \subseteq owl:Thing \wedge_{=1} org:description.sapl:Container \quad \text{(A.6)}$$

9.2 Modelling Complex Functions

Figure 10



In the next step the definition of multi attribute functions itself is in the focus. Such functions map a set of RDF statements to a real valued scalar. Is the scalar interpreted as a utility valuation those complex functions can be used to gain information about the level of satisfaction with a state of the world.

Multi attribute functions are modelled not as instances of a class, but as subclasses of the class *org:ComplexFunction*. This is because such a function is used to be matched not only against one single state, but to specify a class of states to whom it can by applied to; i.e. it specifies which individual states are instances that can be evaluated. The specification, which states satisfy the condition of evaluation, is done by a container connected to the class *org:ComplexFunction* upon the predicate *sapls:Restriction*. The container hosts those statements that have to be part of the state description to be evaluated by the complex function. The namespace “sapls” refers to sapl – schema; thus, the specification of triple patterns and “Restriction” to the notion that the statements hosted in the restriction-container are the minimal requirement for matching the schema (Terziyan, 2008). Consider the graph in the upper part of figure 10 as an example. The description connected with the

state in figure 9 satisfies the restriction, since all required statements, in the schema, are part of the description.

$$org:ComplexFunction \subseteq owl:Thing \wedge_{=1} sapls:\mathrm{Re}striction.sapl:Container \\ \wedge_{=1} org:has\mathrm{Associated}Functions.sapl:Container \quad \text{(A.7)}$$

However, such a definition of a schema allows only to search for those states for which the Complex Function applies, but does not contain any information about how to valuate the statements in the description of a state. Thus, a further container is connected, to org:ComplexFunction, by the predicate org:hasAssociatedFunctions (axiom 7). This container has three purposes; first, those parts of the description are identified that are attributes; second, the identified attributes are connected to a real-value number upon the predicate org:hasWeight to determine their relative weight; third, the identified attributes are connected to an instance of org:Function, as introduced in the pervious section, that is used for their valuation. Consider the example in the lower part of figure 10.

9.3 Evaluating Complex Functions

Containers, such as those used for the description of states as well as those used in the restriction schema of Complex Functions, are nested graphs; structural identical to the ones used in examples in paragraph 7.2. Therefore, it is possible to treat the restriction container of a Complex Function as the IF part of a rule which quires for descriptions of states for which the Complex Function applies. Consequently, the first part of the evaluation formula is a rule triggered by a org:ComplexFunction in connection with its schema and using this schema then as part of its own antecedent.

However, as outlined above, the traditional satisfy relation between a set of statements and an antecedent of a rule is no longer sufficient, but only a necessary condition, additional information for the calculation of the valuation of a graph has to be taken into account. Thus, in the next step the variables in the container hosting the functions, of a Complex Function, are connected to the variables in its schema part, by string matching of its names. This allows to derive for each variable (attribute), its associated weight and valuation function. The body of the evaluation rule triggers then the evaluation of all

attributes according to the appropriate function, of the type org:Function as discussed in section 8. The value calculated by the function, multiplied with the weight of the attribute, can be interpreted as the valuation that a single attribute contributes to the overall valuation of a Complex Function. In the last step of the sum of weighted individual valuations, of attributes, is calculated. The independent valuation of each attribute is in line with the additive utility model discussed in paragraph 4.2. The formula below shows the central aspects, as discussed; for the complete formula used in the prototype see appendix A.4.

```
{{      sapl:I :calculate {?state ?complexFunction *}.
        ?complexFunction rdfs:subClassOf org:ComplexFunction;
                sapls:restriction ?schema;
                org:hasAssociatedFunctions ?functionContainer.
        ?state ?perdic {?schema sapl:is sapl:true }.
        ?functionContainer sapl:hasMember {
        ?attribute org:hasFunction  ?atribut_function;
                org:hasWhight ?atribut_weight }.
        {?schema sapl:hasMember {?match * *}. ?attribute = ?match}
sapl:or
        {?schema sapl:hasMember {* * ?match}. ?attribute = ?match}.
        ?NumberOfAttributes_1 sapl:count ?attribute.
        } sapl:All ?attribute.
} => {
        sapl:I :calculate {?attribute ?atribut_function *}.
        {       sapl:I :calculate {?attribute ?atribut_function ?valuation}.
        ?weightValuation sapl:expression "?atribut_weight*?valuation".
        }=>{    ?attribute :hasValue ?weightValuation
        }.
        {       * :hasValue ?weightValuation.
                ?NumberOfAttributes_2 sapl:count ?weightValuation.
                ?NumberOfAttributes_2 = ?NumberOfAttributes_1.
                ?result sapl:sum ?weightValuation.
```

```
}=>{
sapl:I :calculate {?state ?complexFunction ?result}.
?state org:hasValuation ?result. }}.
```

The aggregation of the values for each attribute to the overall result and the adding of the result are done in the THEN-part. Analogue to the previous section modelling Complex Functions is a relation between a concrete description of a org:Thing, a org:ComplexFunction and a real-valued scalar, to which the later maps the former. In addition to the adding of the result into the container sapl:I :calculate, the thing evaluated upon a Complex Function gets connected to the result of the evaluation by the predicate org:hasValuation. The benefit of such a straight connection becomes visible in the next section.

9.4 Refining Complex Functions

Once a Complex Function is defined it is often needed to further specify either the thing's to which it can by applied to or to select additional attributes. Such a specification of types is done in ontologies via the concept of inheritance (Bechhofer, 2004). The first step is the creation of a subclass of the Complex Function that should be specified. In the next step a container connected to the newly created subclass by the predicate sapls:Restriction, allows to add additional statements to the schema part, that has to be satisfied by instances of org:Thing; thus, to further restrict the scope of the Complex Function. Further attributes can be identified by connecting a container by the predicate org:hasAssociatedFunctions to the new created subclass. Consider the following example S-APL code that continues the example given in figure 10.

```
:colorRequirement rdfs:subClassOf :printReqirement;
    sapls:Restriction {* :hasColorMode ?c};
    org:hasAssociatedFunctions {?c org:hasFunction :f3;
        org:hasWhight "2"}.
```

To ensure that the class :colorRequirement has all the restrictions and functions its super classes have, two rules are needed to infer the class hierarchy. The rule that is necessary to infer the restriction hierarchy can be reused from the Semantics-Based Access Control Reasoner (Terziyan, 2008); the one that is needed to grant that Complex Functions inherit the associated

Functions and weights of the attributes of their super classes is depicted in the following formula; for the complete formula see appendix A.5.

```
{{      ?X org:hasAssociatedFunctions ?own.
        ?X rdfs:subClassOf ?C.
        ?C org:hasAssociatedFunctions ?super.
        ?super sapl:hasMember ?id.
} => {  ?own sapl:hasMember ?super}.
```

10 Contracts

After the introduction of how states of the KB, made up as *org:Thing's*, can be evaluated by *org:ComplexFunction's* to allow the mapping of a state of the world to a real-valued scalar; it is shown in this section how this real-valued scalar, interpreted as a utility valuation, can be used determine the value of actions; and to decide which actions should be made and which not. Since the context of this work is an economic one the actions that will be considered are selling and buying. To establish a mutual beneficial agreement, in a potential cooperative situation between a seller and a buyer, a contract is needed to mark the terms to which both agree. Auctions are such a coordination mechanism to establish mutual beneficial agreements. A statement that indicates that an agent provides something is called offer; a statement indicating the interests in an offer is called bid (McAfee, 1987).

10.1 Auctions

As discussed in paragraph 4.4 auctioning is a mechanism to determine a contract between market principals (agents). In this paragraph the mechanism of the Vickery auction will be deployed to establish a mutual beneficial contract between two agents.

In the first step it has to be determined how to offer something; thus, how to set up an auction. Since agents have to have the ability to determine the value of what the auction is about, the class *org:Auction* is modeled as a subclass of the class *org:Thing*. This has the advantage that the object of the auction, the description what the auction is about, can be valuated upon a Complex Function that acts as Utility Function. A Vickery auction,

as a special kind of auction, is modeled as a subclass of the class auction as formalized in axiom A.8. and A.9. The additional intermediate class auction brings the advantage that additional types of auctions, not considered in this thesis, can be added without changing the ontology.

$$org:Auction \subseteq org:Thing \wedge_{=1} \text{org:hasAuctioneer}.sapl:Agent \quad \text{(A.8)}$$

$$\begin{aligned} &org:Vic\ker yAuction \subseteq org:Auction \wedge_{<2} org:hasDurnation.xsd:\text{int} \\ &\wedge org:hasMin\,\text{Pr}ice.xsd:double \wedge org:hasBid.org:Bid \\ &\wedge_{<2} org:hasWinner.sapl:Agent \wedge_{<2} org:hasMarket\,\text{Pr}ice.xsd:double \end{aligned} \quad \text{(A.9)}$$

$$org:Bid \subseteq_{=1} org:hasAgent.sapl:Agent \wedge_{=1} org:hasValuation.xsd:double \quad \text{(A.10)}$$

$$org:hasMin\,\text{Pr}ice \subseteq org:hasValuation.xsd:double \quad \text{(A.11)}$$

The predicate org:hasAuctioneer connects an auction with the name of the agent making the offer. The parameter *org:hasDuration* acknowledges the pragmatic assumption of not having a point market, but giving the agents time to response. The parameter *org:hasMinPrice* aids as a mean to ensure that an offering agent does not have to give away something below the valuation upon its utility for it; but bids below the minimal price will not further be considered. In other words, the parameter org:hasMinPrice is the restriction that the Utility Function policy, of the auctioneer, puts on the potential action of selling something. Thus, the datatype property org:hasMinPrice is a sub predicate of org:hasValuation as discussed in the previous section, see Axiom A.11.

10.2 Bidding

After an agent has created an instance of the class *org:VickreyAuction* and connected it with its name, by the predicate *org:hasAuctioneer*, as well as determined the minimal price he is going to accept, in the datatype property org:hasMinPrice, the instance and its connected objects are published to other agents. Since the deployed communication mechanism is not central, it will not further be discussed. However, a schematic illustration is given with figure 11.

An agent can determine if it has received an offer by searching for instances of the Vickrey auctions whose property *org:hasAuctioneer* is different from its own name. If an agent finds such an offer a second evaluation of the auction starts. However this time, not by the Utility Function of the seller, but upon its own Utility Function a valuation is calculated. Thus, in this stage an agent is in the position to compare its own utility valuation, for the thing the auction is about, with the minimal price set by the seller. Likewise to the seller's side the Utility Function policy determines now in which potential trades (actions) the agent is interested in, as well as its maximal willingness to pay.

As in the previous paragraph pointed out (axiom A.9) instances of the class Vickery auction can have an arbitrary number of bids connected by the predicate *org:hasBid*. Whereby each single instance of the class org:Bid connects the name of the bidder with a valuation, in the height it likes to bid, as formalized by axiom A.10. Therefore, an agent makes a bid by creating an instance of the class org:Bid and connecting it together with its name and its utility valuation to the auction. The next formula formulizes this by adding a bid to instances of a Vickery auction in the height of the utility valuation, if one exists. Since the Vickrey auctions have the property of self-revelation, creating a bid for each auction in the height of the evaluated utility valuation is a dominant strategy (see paragraph 4.4). Sending the information back to the auctioneer is again omitted.

```
{       ?auction rdf:type org:VickreyAuction;
        org:hasMinPrice ?minPrice.
        ?auction org:hasValuation ?util.
        ?util > ?minPrice.
}=>{
        ?auction org:hasBid [ org:hasAgent sapl:I; org:hasValuation ?util]
}.
```

Please note that the square brackets, in the object path of the consequence of the rule, are the blank node syntax of N3 to express that there exists something (a bid) that is connected to the statements within the bracket (Berners-Lee, 2006).

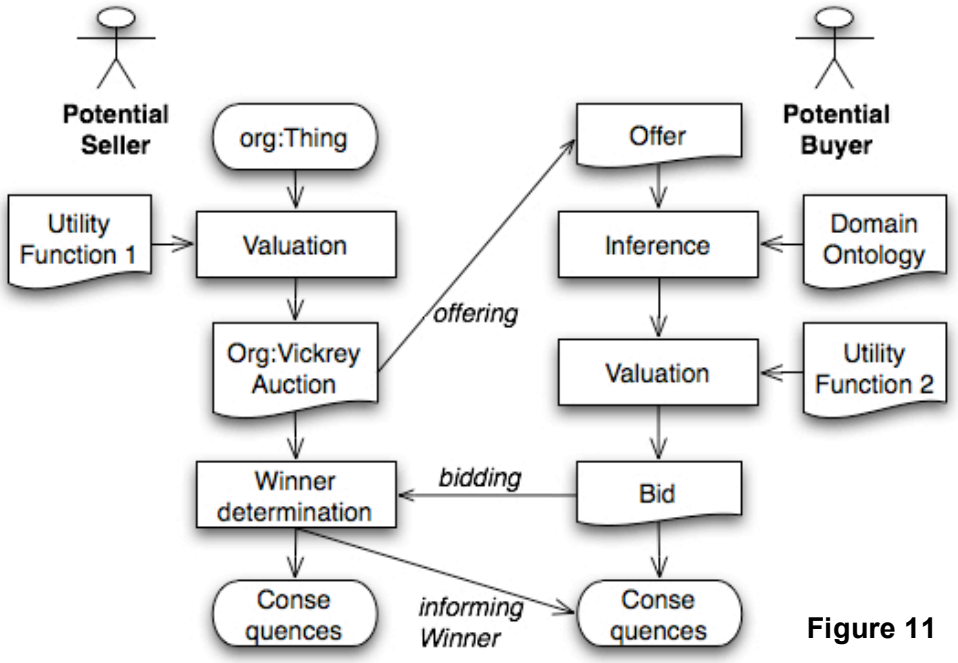


Figure 11

10.3 Reasoning inside a Container

One essential aspect has not been considered so far: RDFS and OWL inference. While the basic rules of entailment are available in the root container (Katasonov, 2008) no approach to the inference with these rules inside containers has been put forward so far (as far as the author is aware). However, such an inference is necessary to allow that a vocabulary, an agent has, can be applied to the description of a bid. Since a detailed theoretical discussion would exceed the limitations of this thesis, only a very practical attempt will be made. The basic idea is that the vocabulary and the statements, to which the vocabulary should be applied to, are located in distinct containers (ontological spaces). Consequently, the RDFS and OWL rules of entailment have to be inferred for the vocabulary container in the first place. Moreover, the deductive cloud, generated upon the OWL and RDFS rules, of the vocabulary in conjunction with the statements in the description container, is added only to the other container and not to the one hosting the vocabulary. This enables the agents to apply a vocabulary to an offer in such a way that the new facts created upon the offer together with the vocabulary are only inside the description container of the offer. The following formula depicts as an example the RDFS rule of entailment rule9 (Hayes, 2004); for

the formulas necessary to infer other rules of entailment have a look in appendix A.7.

```
{       ?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?A rdfs:subClassOf ?B}.
        ?c2 sapl:hasMember {?S rdf:type ?A}.
} => { ?c2 sapl:hasMember {?S rdf:type ?B}}
```

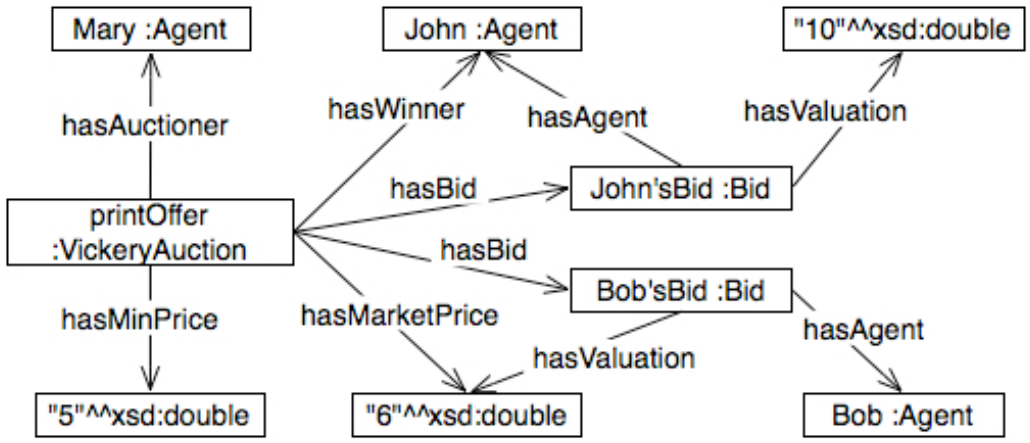
10.4 The Evaluation

In the last step of an auctioning process the winner and the price to be paid have to be determined. As outlined in paragraph 4.4 the highest bidder wins a Vickery auction; however, only the costs that are put on the other participants are charged. The parameter org:*hasDuration* determines when the auction will be closed and the evaluation can begin after this point in time has passed. If bids are made the formula determines in the first step which is the top bid; in the second step which is the second highest bid; and in the last step who has made the top bid. In the consequent graph the property *org:hasWinner* and the datatype property org:hasMarketPrice are added to the auction; the first connecting the name in the top bid, the second connecting the valuation in the second highest bid. The cases of only one bidder or only one bidder bids above the minimal required bidding price are omitted in the following formula; however, made accessible to the interested reader in appendix A.6.

```
{{      ?auct rdf:type org:VickreyAuction;
                org:hasMinPrice ?minprice;
                org:hasBid [org:hasAgent ?agent;
                        org:hasValuation ?bidPrice, ?SecBidPrice].
                ?SecBidPrice < ?bidPrice.
                ?SecMaxBidPrice sapl:max ?SecBidPrice.
        ?auct org:hasBid [org:hasAgent ?winner;
        org:hasUtility ?SecMaxBidPrice].
}=>{
        ?auct org:hasWinner [org:hasAgent ?winner;
                org:hasValuation ?SecMaxBidPrice]}.
```

Figure 12 shows an exemplary situation, utilizing the concepts introduced so far. Agent Mary launches the auction *:printOffer* with the minimal pricing of 5 Utils. Agent John receives the offer, evaluates it, with its Utility Function upon the value of 10 Utils, and places his bid accordingly. The same applies to agent Bob; however, his Utility Function results only with a valuation in the height of 6 Utils. After both agents have made their bids, the auctioneer Mary determines the winner John and the price to be paid in the amount of 6 Utils, leading to an overall increase in the utility for the agents in the amount of 1+4=5 Utils.

Figure 12



11 Dynamic Allocation

The approach discussed so far enables agents to express a utility for goods and to acquire them. However, the approach runs into a problem if not only one item can be acquired, but multiple ones. In other words, it is possible to express the utility an agent gets from the ownership of a print service, but it remains open how to model that an agent might have a lower benefit from acquiring an additional print opportunity than in the first place. First attempts to deal with a decreasing utility of an additional unit of a good are made by Cramer (1728) and Daniel Bernoulli (1738), as an attempt to solve the St. Petersburg, building the fundament of what is later known as Marginal Revolution.

While in the previous section the central concern was to whom a single good should be allocated, to reach a global optimum, the question to answer in this section is: How to distribute the proportions of a dividable good in such a way that a global optimum is reached? To achieve this the already introduced concept of auctions will be changed in two ways: First, the minimal bidding price and the bidding price itself will not longer be derived upon the valuation by the Utility Function, but by the difference a loss or gain (i.e. Border-use), of the org:Thing the auction is about, would bring. Second, not only one auction will be considered, but also a series of auctions will be used to achieve an optimal allocation. In the first paragraph of this section the makeup of the current KB, for the change an auction would bring to it and for the resulting hypothetical state of the KB, is in the focus. In the next paragraph the calculation of the value of the change from the current to the hypothetical KB is in the centre. In the last paragraph a small simulation, as proof of concept, is conducted.

11.1 Locking into the Future

In a first step the class org:Thing has to be examined. Up to yet, the class is only used to describe the configuration of one single and static item. However, no theoretical constraints enforce this. Moreover, the class is defined only as a subclass of owl:Thing that has to have a container in the subject part connected to it (axiom A.6). In other words, the class org:Thing is the parent class of all instances that have a container in their subject-path. Thus, org:Thing can be used to host the agents representation of the state of the world in a very general way. Since this thesis has an economic context, the state of the world is determined for an agent by its ownership and the class org:Thing is used to represent this ownership. Consider the next statement as a minimal example of such an economic KB.

:myKb org:hasDescription {sapl:I ex:haveEnergy "4.8"^^xsd:double}.

In the next step three types of such an economic Knowledge Base have to be distinguished: current KB, hypothetical KB, and a KB representing the difference between the former and the later.

The representation of the amount of things currently controlled by the agent is modelled by the class org:CurrentKb, subclass of org:Thing; whereby each agent, participating in the dynamic allocation process described here, has to have exactly one instance of org:CurrentKb.

$$org:CurrentKb \subseteq org:Thing \quad \text{(A.12)}$$

$$org:\mathrm{HypotheticalKb} \subseteq org:Thing \quad \text{(A.13)}$$

Neglecting eventual productive capabilities of agents the change to an instance of the class org:CurrentKb is reduced to exchange with other agents; thus, to auctioning as it is discussed in the previous section. Likewise, the change to the current KB will be represented by instances of the class org:Auction.

However, to prohibit that an agent gives something away below its own valuation for it, the preference for the thing an auction is about has to be calculated. In other words, the difference a change in the current KB would cause to the utility level of an agent is needed. To accomplish this a hypothetical KB has to be constructed. To identify the hypothetical KB as such the class org:HypotheticalKb is introduced (axiom A.13), analogously to the class org:CurrentKb.

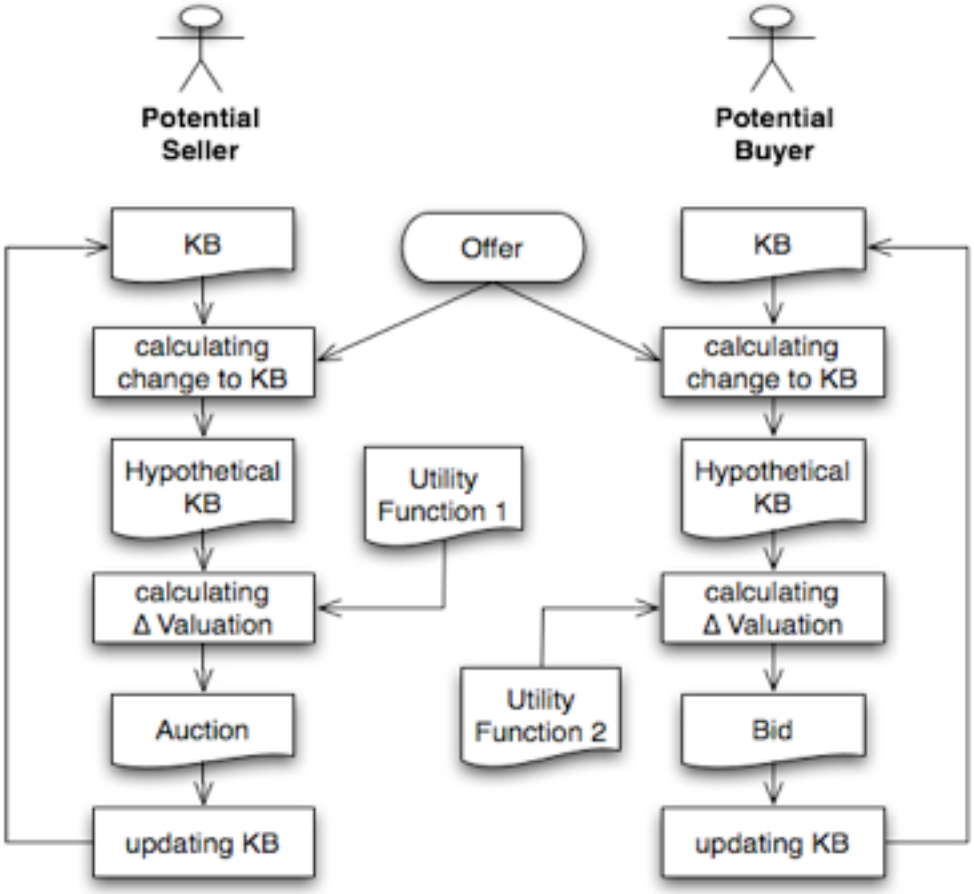
11.2 Determining the Value Of Change

The calculation of the value of a possible action (e.g. org:VickreyAuction), leading to a change in the KB, can be reduced to the calculation of the difference between the current and the hypothetical KB, as well as the construction of the hypothetical KB.

The construction of a hypothetical KB, for a given current KB and a given change to it, depends on the schema of the KB (i.e. the application ontology). Thus, no general formula can be given, but a continuation of the example KB given in the previous paragraph illustrating the very steps of the construction of a hypothetical KB. In the head of the formula the current KB as well as the change, which will potentially be applied to it, are bound to variables. In the next step the difference is calculated. In the body of the formula the

hypothetical KB is constructed and evaluated along with the current KB by the Utility Function of the agent.

Figure 13




```
{       ?currentKb org:Description {* ex:haveEnergy ?eCur};
                rdf:type org:CurrentKb.
        ?change org:hasDescription {* ex:haveEnergy ?eChange};
                rdf:type org:Auction
        ?eHyp sapl:expression "?eCur-eChange".
}=>{
        _:hypKB org:hasDescription
                        {sapl:I ex:haveEnergy ?eHyp}.
        sapl:I org:calculate {?currentKb ex:UtilityFunction *}.
        sapl:I org:calculate {_:hypKB ex:UtilityFunction *}.
}.
```

Likewise, this formula has to be implemented also for the agent on the buyer's side. However, constructing the hypothetical KB on the sum of the change, a won auction would bring, to the current KB.

The calculation of the value of the change to the agent is the calculation of the difference in valuation between the current KB and the hypothetical, as depicted in formula below.

```
{       sapl:I org:calculate {?currentKb ex:UtilityFunction ?currentUtility}.
        sapl:I org:calculate
                {_:hypKB ex:UtilityFunction ?hypotheticalUtility}.
        ?currentKb rdf:type org:CurrentKb.
        ?hypotheticalValuation sapl:expression
                "?hypotheticalUtility-?currentUtility".
        ?change rdf:type org:Auction.
}=>{
        ?change org:hasValuation ?hypotheticalValuation.
}.
```

In contrast to the construction of the hypothetical KB, the calculation of the value of the change requires no differentiation between the buyers and sellers side and is domain independent; for the complete version used in the prototype, see appendix A.8.

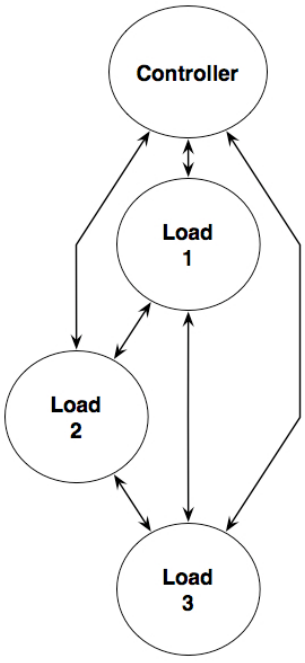
11.3 An Application Example

Up to yet, all basic components of a market in a Multi-Agent system have been introduced. However, an application example, close to the demands of real-world situations, remains to be given. In this paragraph it is presented how the approach can be used to handle the application of power load management. The reader is asked to keep in mind that this is only a proof of the method presented in this thesis. Consequently, only mathematical formulas are given that are relevant for the properties of the application, but no formulas are given for a specific domain.

Any device that consumes electric energy, such as electric heating systems or streetlights, is considered as a load. Load management refers to the concept of controlling the amount of load a device receives, to achieve an efficient use of energy. For an in depth review of the concept of load management, and a differentiation between direct an indirect load management, see (Ygge, 1996).

There are a number of reasons why the approach put forward in this thesis is interesting in the area of load management. On the one hand side, since the information is distributed through the system, and not transmitted to a central point, there is a potential gain in reduced communication as well as in stability. On the other hand side, the task of load management is computational complex and a decentralized approach can utilize the inherent computational power of each node in the network. Furthermore, from an engineering point of view a system of distributed agents, instead of a centralized one, enables to add, delete and modify loads without changing the entire system. Last but not least, due to the W3C standards, followed in this thesis, the approach can easily be integrated into an existing infrastructure.

Figure 14



The market is designed in such a way that one agent represents one controllable load. The need of a load for a share of the globally constraint resource (energy), is expressed by the Utility Functions of the agent. Conforming to standard utility theory the Utility Functions are concave so that the first derivative (i.e. the bidding price) decreases with increased share of the resource (Oberender, 2004 p.169). Thus, the second derivative of the Utility functions is negative.

Beginning from the Utility Functions and the initial distribution of the resource the market mechanism settles the distribution of the energy.

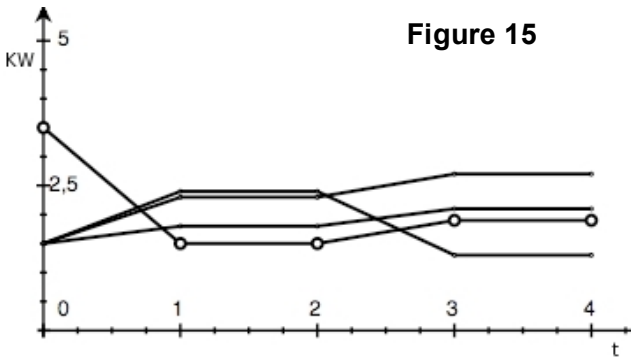
Besides the agents representing loads, a Controller agent is used to provide a way to manipulate the loads in order to make the loads to behave in a certain way, as suggested by Ygge (1996). Thus, the Controller agent can be thought

of as an instance to manage the whole system. Figure 14 depicts the used topology.

In an auction, all agents receive an offer from the auctioning agent and send a bid back to the designated auctioneer. In the next step a reallocation is computed as discussed in section 10. The roles auctioneer and bidder are not static, but an agent enacting the auctioneer role schedules auctions as long as in the last auction at least one bidder submitted a bid above the minimal bidding price; has no agent submitted such a bid, another agent gets assigned with the role the auctioneer.

The goal is to maximize the global utility U_{glob} of all agents that they have from the resource energy e. Whereby the global utility is the sum of the utilities the agents have $u_{glob}(e_1,...,e_n) = u_1(e_1) + ... + u_n(e_n)$, and the sum of energy is the globally constraint by $E = e_1 + ... + e_n$. The global utility maximum is reached if the so-called Kuhn-Tucker conditions are fulfilled; thus, if all first derivatives of the agents Utility Functions are equal to a shared value (Oberender, 2004 pp. 230). In other words, an optimal distribution of energy is reached if all agents express the same need (bidding price) for an additional unit of the resource.

Figure 15



In figure 15, the performance of a simulation is shown. Each load agent is assigned with initial share of the available energy of 1.5 KW. The Controller agent (depicted in the figure by the graph with circles) is assigned with 3.5 KW. From such an initial, suboptimal, distribution equilibrium is reached after one round. A round is finished after each agent has enacted the auctioneer role. In the second round a load agent tries to reduce its load by changing its Utility Function and consequently having a lower valuation for a share of the resource. Again, the system settles in the next round.

If the price of the energy, expressed by the agents, directly corresponds to the real price of resource it can immediately be controlled how much a change in the demand of an agent or supply costs. This is in particular useful for the management of such a system (Ygge, 1996).

12 Conclusion

In this last section the achievements and shortcomings of the presented approach are reviewed and possible solutions are shortly examined.

12.1 Achievements, Limitations and further Research

In this thesis the author gave a structured overview on policies as well as mathematical models for their application. Furthermore, a novel approach for their application on the selection of the actions (i.e. trades) those are not only good enough, but are also the best action for a given choice set (i.e. bids). The author claims that the approach put forward has several advantages compared to current methods (as cited in paragraph 2.2): First, it is not bound to any specific domain. Second, the selection of attributes, relevant to the agent, is done by the agent and not determined a priori by a second party. Third, the agents are not confronted with static prices, but with prices emerging as a result of supply and demand, allowing to identify potential bottlenecks and to monitor the overall performance of the system. This is in particular useful since ultimately all activities are bound to economic exchanges (Adelsberger, 2000), and the problem of relating cause and effect is easier if the interdependence of supply and demand is made explicit.

However, the applicability of the approach is also restricted. First, in the immediate scope of the thesis are only rational (i.e. for both parties beneficial) trades. Second, the approach has just been applied to situations with only two commodities, (i.e. a good and money in terms of Utils). Third, the allocation has been carried out by sequential auctions. Thus, only one item is auctioned at a time. The author considers however, that preferences of an agent towards bundles (i.e. combinations of items) have not been covered as the biggest limitation. A typical example, for bundles, is a computer and a monitor; whereby the valuation for each item is lower then for both together. However, several authors have addressed the problem of bundles.

The central difficulty is that to determine the valuation for an individual item; the bidder needs to know what parts of the bundle it will receive in later auctions. This requires speculations about bids of other agents, because they affect what the agent is going to receive. Moreover, what other agents bid depends again on what they believe about their competitors. This counter speculations lead to NP-hard problems in terms of computational costs (Fujishima, 1999). While the computational overhead, in sequential auctions, cannot be resolved various approaches have been put forward in order to fix the inefficient allocations that emerge from the uncertainties. One approach is to install an after market for enabling the bidders to exchange things after the main auction has been closed. Such an approach can correct some inefficiency. However, it is not guaranteed that a Pareto efficient allocation is reached and it involves a large number of exchanges (Rothkopf, 1998). Another approach, discussed by Sandholm and Lesser (1995; 1996), is to allow agents to retract their bids if they do not get the desired combinations. In the case of a retraction, the item gets auctioned again. However, the possibility of retraction can lead to gaming, in particular if a bidder believes that it can get the item for a lower price. A third approach is to allow bidders to place bids on combinations of items as discussed by Vries (2003). While combinatorial auctions free the agents from the look-ahead, they require a central auctioneer to determine an optimal allocation. Typically this is a non-trivial NP-hard task. Sandholm (2001) put forward promising work to tackle the computational complexity.

12.2 Acknowledgements

I want to thank my friends and colleagues at the Industrial Ontologies Group as well as the University of Jyväskylä. Without their support, this work would not have been possible.

Appendix

Appendix A.1

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix org: <http://www.example.com/org#>.
{{      sapl:I sapl:want {sapl:I :calculate {?a ?f *}}.
        ?f  rdf:type org:DiscreteFunction;
                org:hasPoint ?p.
        ?p org:hasAttribute ?ar;
                org:hasUtility ?u.
        ?ar = ?a.
} >> {
        sapl:I :calculate {?a ?f ?u}.
}} sapl:is sapl:Rule.
```

Appendix A.2

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix org: <http://www.example.com/org#>.
{{
        sapl:I sapl:want {sapl:I :calculate {?a ?f *}}.
        ?f rdf:type org:LinearFunction.
} >> {{
                ?f      org:hasPoint ?1p;
                        org:hasPoint ?2p.
                ?1p org:hasAttribute ?1v.
                ?2p org:hasAttribute ?2v.
                ?n sapl:expression "?a-?1v".
                ?n >= 0.
                ?t sapl:min ?n.
                ?n = ?t.
                ?b sapl:expression "?2v-?a".
                ?b > 0.
                ?r sapl:min ?b.
                ?b = ?r.
        } => {{
                        ?helper1 org:hasAttribute ?1v;
                                org:hasUtility ?1u.
                        ?helper2 org:hasAttribute ?2v;
                                org:hasUtility ?2u.
                        ?u  sapl:expression  "?1u+(?a-?1v)*((?2u-?1u)/(?2v-
?1v))".
                } =>{
                        sapl:I :calculate {?a ?f ?u}
}}}} sapl:is sapl:Rule.
```

Appendix A.3

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix org: <http://www.example.com/org#>.
{{{     sapl:I sapl:want {sapl:I :calculate {?a ?f *}}.
        ?f rdf:type org:PolynomialFunction;
                org:hasTerm ?p.
        ?p org:hasWeight ?w;
                org:hasExponent ?e.
        ?x sapl:count ?p.
        } sapl:All ?p.
} >> {
        {
                ?u sapl:expression "?w*pow(?a,?e)"
        }=>{
        org:PolynomialFunction :archive { ?a ?f {?p :hasResult ?u}}.
        }.
        {
        org:PolynomialFunction :archive { ?a ?f {* :hasResult ?zu}}.
                ?y sapl:count ?zu.
                ?y = ?x.
                ?h sapl:sum ?zu.
        }=>{
                sapl:I :calculate {?a ?f ?h}.
}       }       } sapl:is sapl:Rule.
```

Appendix A.4

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix sapls: <http://www.ubiware.jyu.fi/sapl_schema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix org: <http://www.example.com/org#>.
{       {sapl:I sapl:want {sapl:I :calculate {?state ?complexFunction *}}.
        } => {{{
        ?complexFunction rdfs:subClassOf org:ComplexFunction;
                sapls:restriction ?schema;
                org:hasAssociatedFunctions ?functionContainer.
        ?state ?perdic {?schema sapl:is sapl:true }.
        ?state != ?complexFunction.
        sapl:I sapl:doNotBelieve {?complexFunction rdfs:subClassOf
?state}.
        sapl:I sapl:doNotBelieve {?state org:hasUtility {* org:upon ?P}}.
        ?functionContainer sapl:hasMember {
        ?attribute org:hasFunction  ?atribut_function.
        ?atributeHelp org:hasWhight ?atribut_whight.
                        }.
        ?atributeHelp = ?atribute.
        {?schema sapl:hasMember {?match * *}. ?attribute = ?match}
sapl:or
        {?schema sapl:hasMember {* * ?match}. ?attribute = ?match}.
        ?NumberOfAttributes_1 sapl:count ?attribute.
        } sapl:All ?attribute.
} => {sapl:I sapl:want {sapl:I :calculate {?attribute ?atribut_function *}}.
        {       sapl:I :calculate {?attribute ?atribut_function ?valuation}.
        ?weightValuation sapl:expression "?atribut_whight*?valuation".
        }=>{    org:ComplexFunction :archive {?state ?complexFunction
{?attribute :hasValue ?weightValuation}}.
        }.
        {       * :hasValue ?weightValuation.
```

```
                ?NumberOfAttributes_2 sapl:count ?weightValuation.
                ?NumberOfAttributes_2 = ?NumberOfAttributes_1.
                ?result sapl:sum ?weightValuation.
        }=>{
        sapl:I :calculate {?state ?complexFunction ?result}.
        ?state org:hasValuation ?result.
}}}} sapl:is sapl:Rule.
```

Appendix A.5

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix org: <http://www.example.com/org#>.
{{{     ?X org:hasAssociatedFunctions ?own.
        ?X rdfs:subClassOf ?C.
        ?C org:hasAssociatedFunctions ?super.
        ?super sapl:hasMember ?id.
        sapl:I sapl:doNotBelieve {?own sapl:hasMember ?super}.
} => {
        ?own sapl:hasMember ?super.
} sapl:is sapl:Rule }.
```

Appendix A.6

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix org: <http://www.example.com/org#>.
{{      ?auct rdf:type org:VickreyAuction;
              org:hasDuration ?duration;
              org:hasMinPrice ?minprice;
              org:hasStartTime ?starttime;
              org:hasBid [ org:hasAgent ?winningAgent;
org:hasValuation ?bidPrice].
        {?auct org:hasDescription {?object sapl:is sapl:true}} sapl:or
{?auct org:hasDescription ?object}.
        sapl:Now sapl:is ?now.
        ?now > ?starttime+?duration+400.
        sapl:I sapl:doNotBelieve {?auct org:hasEndTime *}.
        ?bidPrice >= ?minprice.
        ?numberOfBids sapl:count ?winningAgent.
        ?numberOfBids < 2.
}=>{
        ?auct org:hasEndTime ?now.
        ?auct org:hasWinner ?winningAgent;
              org:hasMarketPrice ?minprice;
              org:numberOfBids ?numberOfBids.
}
{       ?auct rdf:type org:VickreyAuction;
              org:hasDuration ?duration;
              org:hasMinPrice ?minprice;
              org:hasStartTime ?starttime;
              org:hasBid ?1bNode, ?2bNode, ?3bNode.
        {?auct org:hasDescription {?object sapl:is sapl:true}} sapl:or
{?auct org:hasDescription ?object}.
        sapl:Now sapl:is ?now.
```

```
        ?now > ?starttime+?duration+400.
        sapl:I sapl:doNotBelieve {?auct org:hasEndTime *}.
        ?1bNode org:hasAgent ?winningAgent;
                org:hasValuation ?1MaxBidPrice.
        ?2bNode org:hasAgent ?2ag; org:hasValuation ?2MaxBidPrice.
        ?1MaxBidPrice > ?2MaxBidPrice.
        ?SecMaxBidPrice sapl:max ?2MaxBidPrice.
        ?numberOfBids sapl:count ?3bNode.
        ?numberOfBids > 1.
        ?1MaxBidPrice >= ?minprice.
}=>{
        ?auct org:hasWinner ?winningAgent;
                org:numberOfBids ?numberOfBids.
        ?auct org:hasEndTime ?now.
        {?SecMaxBidPrice > ?minprice} -> {
        ?auct org:hasMarketPrice ?SecMaxBidPrice.
                }; sapl:else {
                ?auct org:hasMarketPrice ?minprice.
}}} sapl:is sapl:Rule.
```

Appendix A.7

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

Entailment Rule: rdfs2

```
{{{{    ?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdfs:domain ?C}.
        ?c2 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve {?c2 sapl:hasMember {?S rdf:type ?C}}.
        } sapl:All ?S } sapl:All ?C.
} => { ?c2 sapl:hasMember {?S rdf:type ?C}}} sapl:is sapl:Rule.
```

Entailment Rule: rdfs3

```
{{{{    ?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdfs:range ?C}.
        ?c2 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve {?c2 sapl:hasMember {?O rdf:type ?C}}.
        } sapl:All ?O} sapl:All ?C.
} => { ?c2 sapl:hasMember {?O rdf:type ?C}}} sapl:is sapl:Rule.
```

Entailment Rule: rdfs5

```
{{{{    ?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember
        {?Q rdfs:subPropertyOf ?R. ?P rdfs:subPropertyOf ?Q}.
        sapl:I sapl:doNotBelieve
        {?c1 sapl:hasMember {?P rdfs:subPropertyOf ?R}}.
```

```
        } sapl:All ?P} sapl:All ?R.
} => {?c1 sapl:hasMember {?P rdfs:subPropertyOf ?R}}
} sapl:is sapl:Rule.
```

Entailment Rule: rdfs7

```
{{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdfs:subPropertyOf ?R}.
        ?c2 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve {?c2 sapl:hasMember {?S ?R ?O}}.
        } sapl:All ?S} sapl:All ?R} sapl:All ?O.
} => { ?c2 sapl:hasMember {?S ?R ?O}}} sapl:is sapl:Rule.
```

Entailment Rule: rdfs9

```
{{{{    ?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?A rdfs:subClassOf ?B}.
        ?c2 sapl:hasMember {?S rdf:type ?A}.
        sapl:I sapl:doNotBelieve {?c2 sapl:hasMember {?S rdf:type ?B}}.
        } sapl:All ?S} sapl:All ?B.
} => { ?c2 sapl:hasMember {?S rdf:type ?B}}} sapl:is sapl:Rule.
```

Entailment Rule: rdfs11

```
{{{{    ?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember
        { ?B rdfs:subClassOf ?C. ?A rdfs:subClassOf ?B} .
        sapl:I sapl:doNotBelieve
        {?c1 sapl:hasMember {?A rdfs:subClassOf ?C}}.
        } sapl:All ?A} sapl:All ?C.
} => { ?c1 sapl:hasMember {?A rdfs:subClassOf ?C}}} sapl:is sapl:Rule.
```

OWL Entailment Rule: Symmetric property

```
{{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdf:type owl:SymmetricProperty}.
        ?c2 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve { ?c2 sapl:hasMember {?O ?P ?S}}.
        } sapl:All ?P} sapl:All ?S } sapl:All ?O.
} => {?c2 sapl:hasMember {?O ?P ?S}}.
{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdf:type owl:SymmetricProperty}.
        ?c1 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve { ?c1 sapl:hasMember {?O ?P ?S}}.
        } sapl:All ?P} sapl:All ?S } sapl:All ?O.
} => {?c1 sapl:hasMember {?O ?P ?S}}} sapl:is sapl:Rule.
```

OWL Entailment Rule: Inverse property

```
{{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P owl:inverseOf ?Q}.
        ?c2 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve { ?c2 sapl:hasMember {?O ?Q ?S}}.
        } sapl:All ?O} sapl:All ?Q} sapl:All ?S.
} => {?c2 sapl:hasMember {?O ?Q ?S}}.
{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P owl:inverseOf ?Q}.
        ?c1 sapl:hasMember {?S ?P ?O}.
        sapl:I sapl:doNotBelieve { ?c1 sapl:hasMember {?O ?Q ?S}}.
        } sapl:All ?O} sapl:All ?Q} sapl:All ?S.
} => {?c1 sapl:hasMember {?O ?Q ?S}}} sapl:is sapl:Rule.
```

OWL Entailment Rule: Transitive property

```
{{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdf:type owl:TransitiveProperty}.
        ?c2 sapl:hasMember {?S ?P ?X. ?X ?P ?O}.
        sapl:I sapl:doNotBelieve {?c2 sapl:hasMember {?S ?P ?O}}.
        } sapl:All ?S} sapl:All ?P} sapl:All ?O.
} => {?c2 sapl:hasMember {?S ?P ?O}}.
{{{{?c1 :appliesTo ?c2.
        ?c1 sapl:hasMember {?P rdf:type owl:TransitiveProperty}.
        ?c1 sapl:hasMember {?S ?P ?X. ?X ?P ?O}.
        sapl:I sapl:doNotBelieve {?c1 sapl:hasMember {?S ?P ?O}}.
        } sapl:All ?S} sapl:All ?P} sapl:All ?O.
} => {?c1 sapl:hasMember {?S ?P ?O}}.
} sapl:is sapl:Rule.
```

Appendix A.8

```
@prefix sapl: <http://www.ubiware.jyu.fi/sapl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix org: <http://www.example.com/org#>.
{{      sapl:I org:calculate {?currentKb ?utilityFunction ?currentUtility}.
        sapl:I org:calculate
                {?hypotheticalKb ?utilityFunction ?hypotheticalUtility}.
        ?currentKb rdf:type org:CurrentKb.
        ?hypotheticalKb rdf:type org:HypotheticalKb.
        ?utilityFunction rdfs:subClassOf org:ComplexFunction.
        ?hypotheticalValuation sapl:expression
                "?hypotheticalUtility-?currentUtility".
        ?change rdf:type org:Auction.
        sapl:I sapl:doNotBelieve
                {?change org:hasValuation ?hypotheticalValuation}.
}=>{
        ?change org:hasValuation ?hypotheticalValuation.
}} sapl:is sapl:Rule.
```

Bibliography

Adelsberger, H.H. (2000). "Economic Coordination Mechanisms for Holonic Multi-Agent Systems," in 11th International Workshop on Database and Expert Systems Applications; p. 236.

Agarwala, Sudhir; Lamparter, Steffen; Studera, Rudi (2008). "Making Web services tradable A policy-based approach for specifying preferences on Web service properties" in Web Semantics: Science, Services and Agents on the World Wide Web.

Badr, N.; Taleb-Bendiab, A.; Reilly, D. (2004) "Policy-Based Autonomic Control Service" in Proceedings Fifth IEEE International Workshop on Policies for Distributed Systems and Networks.

Bai, Q.; Zhang, M. (2006)."Coordinating Agent Interactions Under Open Environments" in Advances in Applied Artificial Intelligence; Idea Group.

Bearden, Mark; Garg, Sachin; Lee, Woei-jyh (2001). "Integrating goal specification in policy-based management" in 2nd International Workshop on Policies for Distributed Systems and Networks; p. 29–31.

Bechhofer, Sean; Harmelen, Frank van; Hendler, Jim; Horrocks, Ian; McGuinness, Deborah, L.; Patel-Schneider, Peter, F.; Stein, Lynn, Andrea (2004). "OWL Web Ontology Language: Reference" W3C Recommendation; http://www.w3.org/TR/owl-ref/.

Bellifemine, Fabio; Poggi, Agostino; Rimassa, Giovanni (2001). "Developing Multi-agent Systems with JADE" in Intelligent Agents VII Agent Theories Architectures and Languages; p. 42–47; Berlin, Heidelberg: Springer.

Berners-Lee, Tim (1998). "Semantic Web Road map" in W3C DesignIssues; http://www.w3.org/DesignIssues/Semantic.html.

Berners-Lee, Tim (1999). Weaving the Web; San Francisco: Harper.

Berners-Lee, Tim; Hendler, James; Lassila, Ora (2001). "The Semantic Web" in Scientific American; Vol. 284; p. 28-37.

Berners-Lee, Tim (2006). "Notation 3: An readable language for data on the Web" in W3C DesignIssues; http://www.w3.org/DesignIssues/Notation3.html.

Berners-Lee, Tim, Connolly, D., Kagal, L., Hendler, J., Schraf, Y. (2008). "N3Logic: A Logical Framework for the World Wide Web." in Journal of Theory and Practice of Logic Programming (TPLP), Special Issue on Logic Programming and the Web.

Bernoulli, Daniel (1738). "Specimen theoriae novae de mensura sortis" in Commentarii Academiae Scientiarum Imperialis Petropolitanae Vol.5; reprinted in translation as "Exposition of a new theory on the measurement of risk" in Econometrica; Vol.22; p.23–36.

Boella, G; van der Torre, L. (2004). "An Agent Oriented Ontology of Social Reality" in Proceedings of Formal Ontologies in Information Systems conference (FOIS'04); p. 199–209.

Biron, Paul; Permanente, Kaiser; Malhotra, Ashok (2004). "XML Schema Part 2: Datatypes Second Edition" W3C Recommendation; http://www.w3.org/TR/xmlschema-2/.

Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M; Maler, Eve; Yergeau, François (2006). "Extensible Markup Language (XML)" W3C Recommendation; http://www.w3.org/TR/REC-xml/.

Brickley, Dan; Guha R.V. (2004). "RDF Vocabulary Description Language 1.0: RDF Schema"; W3C Recommendation. http://www.w3.org/TR/rdf-schema/.

Brussel, H.; Wyns, J.; Valckenaers, P.; Bongaerts, L.; Peeters. P. (1998). "Reference architecture for holonic manufacturing systems" in Computers in Industry; Vol. 37; p. 255–274.

Chandra, A.; Gong, W.; Shenoy, P; (2003). "Dynamic Resource Allocation for Shared Data Centres Using Online Measurements" in Joint International Conference on Measurement and Modelling.

Charlton, P.; Cattoni, R.; Potrich, A.; Mamdani, E. (2000). "Evaluating the FIPA Standards and Its Role in Achieving Cooperation in Multi-Agent Systems" in Proceedings of the 33rd Hawaii International Conference on System Sciences; Vol. 8.

Chevaleyre, Y.; Endriss, U.; Estivie, S.; Maudet. N. (2004). "Multiagent resource allocation with k-additive utility functions" in Proceedings of DIMACS-LAMSADE Workshop on Computer Science and Decision Theory.

Clarke, E. H. (1971). "Multipart pricing of public goods." in Public Choice; Vol. 11; p. 17–33.

Cramer, Gabriel (1728). "Exposition of a New Theory on the Measurement of Risk," in a Letter from Cramer to Nicholas Bernoulli; translated into English by Louise Sommer; Econometrica Vol. 22; p. 23–36.

Das, R.; Kephart, J.O. (2007). "Achieving self-management via utility functions." In IEEE Internet Computing; Vol. 11-1; p. 40–48.

Dias, M. B.; Stentz, A. (2000). "A free market architecture for distributed control of a multirobot system." in 6th International Conference on Intelligent Autonomous Systems p. 115–122.

Ding, Li; Zhou , Lina; Finin , Tim; Joshi, Anupam (2005). "How the Semantic Web is Being Used: An Analysis of FOAF Documents" in Proceedings of the 38th Hawaii International Conference on System Sciences.

Dixit, Avinash; Skeath, Susan (1999). Games of Strategy. New York: Norton.

Dulay, N.; Lupu, E.; Sloman, M.; Damianou, N.(2001). "A policy deployment model for the Ponder language" in Integrated Network Management Proceedings, IEEE/IFIP International Symposium; p.529–543.

Dürst, Martin; Freytag, Asmus (2007). "Unicode in XML and other Markup Languages" W3C Technical Report; http://www.w3.org/TR/unicode-xml/.

Eiter, Thomas; Lukasiewicz, Thomas; Schindlauer, Roman; Tompits, Hans (2004). "Well-Founded Semantics for Description Logic Programs in the Semantic Web" in Lecture Notes in Computer Science; Vol. 3323; p. 81–97; Berlin, Heidelberg: Springer.

Ermolayev, Vadim; Keberle, Natalya; Plaksin, Sergey; Kononenko, Oleksandr; Terziyan, Vagan (2004). "Towards a Framework for Agent-Enabled Semantic Web Service Composition" in International Journal of Web Services Research; Vol. 3; p. 63–87.

Fallside, David C.; Walmsley, Priscilla (2004). “XML Schema Part 0: Primer Second Edition” W3C Recommendation; http://www.w3.org/TR/xmlschema-0/.

Faratin, P.; Sierra, C.; Jennings, N.R. (1998). “Negotiation Decision Functions for Autonomous Agents” in Robotics and Autonomous Systems; Vol. 24, p. 159–182.

Fellbaum, Christiane (1998). WordNet An Electronic Lexical Database; MIT Press: Cambridge.

Fujishima, Yuzo; Leyton-Brown, Kevin; Shoham, Yoav (1999). “Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches” in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence; p. 548–553.

Gaertner, Dorian; Clark, Keith; Sergot, Marek (2007) “Ballroom etiquette: A Case Study for Norm-Governed Multi-Agent Systems” in Coordination, Organizations, Institutions, and Norms in Agent Systems II; p. 212–226.

Gangemi, Aldo; Mika, Peter (2003). “Understanding the Semantic Web through Descriptions and Situations” in On The Move to Meaningful Internet Systems; Berlin, Heidelberg: Springer.

Gou, L.; Luh, P.; Kyoya Y. (1998). “Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation.” in Computers in Industry; Vol. 37; p. 213–231.

Grosof, B.; Poon, T. (2003). “SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions.” in Proceedings of the 12th World Wide Web Conference; Budapest, Hungary.

Groves, T. (1973). “Incentives in teams.” in Econometrica; Vol. 41; p. 617–631.

Hardin, Garrett (1968). “The Tragedy of the Commons” in Science; Vol. 162; p. 1243 – 1248.

Hayes, Patrick (2004). “RDF Semantics” W3C Recommendation; http://www.w3.org/TR/rdf-mt/.

Heller, B., Herre, H. (2004). “Ontological Categories in GOL” in Axiomathes; Vol. 14-1, p.57–76.

Hendler, James (2001). “Agents and the Semantic Web” in IEEE Intelligent Systems; Vol.16; p. 30–37.

Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. (2004). “SWRL: A semantic web rule language combining OWL and RuleML”; W3C Member submission.

Kagal, L.; Finin, T.; Johshi, A. (2003). “A Policy Language for Pervasive Computing Environment.” in Proceedings of IEEE Fourth International Workshop on Policy for Distributed Systems and Networks.

Kalra, N.; Dias, M.B.; Zlot, R.M.; Stentz, A. (2005). “Market-Based Multirobot Coordination: A Comprehensive Survey and Analysis”; tech. report CMU-RI-TR-05-16, Robotics Institute, Carnegie Mellon University.

Katasonov, Artem; Terziyan, Vagan (2007). “SmartResource Platform and Semantic Agent Programming Language (S-APL)” in Lecture Notes in Computer Science; Vol. 4687; p. 25–36; Berlin, Heidelberg: Springer.

Katasonov, Artem; Terziyan, Vagan (2008). “Semantic Agent Programming Language (S-APL): A Middleware Platform for the Semantic Web” in The IEEE International Conference on Semantic Computing The IEEE International Conference on Semantic Computing; p. 504–511.

Keeney, Ralph L.; Raiffa, Howard; Meyer Richard (1993). Decisions with Multiple Objectives Preferences and Value Tradeoffs. Cambridge University Press.

Kephart, J.O.; Walsh, W.E. (2004). “An artificial intelligence perspective on autonomic computing policies” in Fifth IEEE International Workshop on Policies for Distributed Systems and Networks; p. 3–12.

Koestler, A. (1967). The Ghost in the Machine. Hutchinson & Co: London.

Koivunen, Marja-Riitta; Miller, Eric (2001). “W3C Semantic Web Activity” in Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications; HIIT Publications: Helsinki.

Jennings, Nichola (2000). “On agent-based software engineering” in Artificial Intelligence; Vol. 117, p. 277–296.

Lymberopoulos, L.; Lupu, E.; Sloman, M. (2002). “An adaptive policy based management framework for differentiated services networks” in IEEE Third International Workshop on Policies for Distributed Systems and Networks; Monterey, California.

MacKie-Mason, J. K.; Varian, H. R. (1995). Generalized Vickrey auctions. Technical Report, University of Michigan.

Malone, T. W.; Fikes, R. E.; Grant, K. R.; Howard, M. T. (1988). “Enterprise: A market- like task scheduler for distributed computing.” in The Ecology of Computation; North-Holland, New York.

Manola, Frank; Miller, Eric (2004). “RDF Primer” W3C Recommendation; http://www.w3.org/TR/rdf-primer/.

McAfee, R, Preston; McMillan, John (1987). “Auctions and bidding” in Journal of Economic Literature. Vol. 25, p. 699–738.

McGuinness, Deborah L.; van Harmelen, Frank (2004). “OWL Web Ontology Language Overview” W3C Recommendation; http://www.w3.org/TR/owl-features/.

Mika, P. (2004). "Foundations for Service Ontologies: Aligning OWL-S to DOLCE," in Proc. 13th World Wide Web Conf. (WWW 2004); ACM Press, p. 56–572.

Niles, Ian; Pease, Adam (2001). “Towards a standard upper ontology” in Proceedings of the international conference on Formal Ontology in Information Systems; p. 2–9.

Oberender, Peter; Fehl, Ulrich (2004). Grundlagen der Mikroökonomie: eine Einführung in die Produktions-, Nachfrage- und Markttheorie. München: Vahlen.

Oldham, N.; Verma, K.; Sheth, A.; Hakimpour, F. (2006). “Semantic WS-Agreement Partner Selection” in 15th International World Wide Web Conference (WWW2006).

Rosenfeld, Louis; Morville, Peter (2002). Information Architecture for the World Wide Web; O’Reilly.

Rothkopf, Michael; Peke, Aleksandar; Harstad, Ronald (1998). "Computationally Manageable Combinatorial Auctions" in Management Science; Vol. 44-8, p. 1131–1147.

Russell, Stuart, and Peter Norvig (1995). Artificial Intelligence: A Modern Approach. Berkeley: Prentice Hall.

Sandholm, Tuomas; Lesser, Victor (1995). "Issues in automated negotiation and electronic commerce: Extending the contract net framework." in Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95).

Sandholm, Tuomas (2002) "Algorithm for optimal winner determination in combinatorial auctions" in Artificial Intelligence; Vol. 135; p. 1–54.

Sen, Amartya K.; Williams, Bernard A. O. (1982). Utilitarianism and beyond; Cambridge University Press: New York.

Simon, Jonathan; Smith, Barry (2004) "Using Philosophy to Improve the Coherence and Interoperability of Applications Ontologies: A Field Report on the Collaboration of IFOMIS and L&C" in Proceedings of First Workshop on Philosophy and Informatics Cologne.

Smith, Adam, (1937). An Inquiry into the Nature and Causes of the Wealth of Nations; Random House (Modern Library): New York 1776; Reprint .

Smith, Reid D. (1980). "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver" in IEEE TRANSACTIONS ON COMPUTERS; Vol. 29.

Skylogiannisa, Thomas; Antonioua, Grigoris; Bassiliadesc, Nick; Governatorid, Guido; Bikakisa, Antonis (2007). "DR-NEGOTIATE – A system for automated agent negotiation with defeasible logic-based strategies" in Data & Knowledge Engineering, Vol. 63; p. 362–380.

Tamma, Valentina; Aart, Chris; Moyaux, Thierry; Paurobally, Shamimabi; Lithgow-Smith, Ben; Wooldridge, Michael (2005). "An Ontological Framework for Dynamic Coordination" in The Semantic Web p. 638–652.

Terziyan, Vagan; Katasonov, Artem; Kaykova, Olena; Khriyenko, Oleksiy; Loboda, Oleksiy; Naumenko, Anton; Nikitin, Sergiy (2007). "The Central Principles and Tools of UBIWARE"; Technical Report (Deliverable D 1.1); UBIWARE Tekes Project, Agora Centre, University of Jyväskylä; http://www.cs.jyu.fi/ai/OntoGroup/UBIWARE_Restricted/D1_1.pdf.

Terziyan, Vagan; Katasonov, Artem; Kaykova, Olena; Khriyenko, Oleksiy; Loboda, Oleksiy; Naumenko, Anton; Nikitin, Sergiy (2008). "UBIWARE Platform Prototype v.1.0"; Technical Report (Deliverable D 1.3); UBIWARE Tekes Project, Agora Centre, University of Jyväskylä; http://www.cs.jyu.fi/ai/OntoGroup/UBIWARE_Restricted/D1_3.pdf.

Thomas, Panagiotis; Teneketzis, Demosthenis; MacKie-Mason, Jeffrey K. (2000). "A Market-Based Approach to Optimal Resource Allocation in Integrated-Services Connection-Oriented Networks" in Proceedings of the Fifth INFORMS Telecommunications Conference; Boca Raton: Florida.

Vázquez-Salceda, J.; Dignum, V.; Dignum, F. (2005). "Organizing Multiagent Systems" in Autonomous Agents and Multi-Agent Systems; Vol. 11, p. 307–360.

Vries, Sven; Vohra, Rakesh (2003). "Combinatorial Auctions: A Survey" in INFORMS JOURNAL ON COMPUTING; Vol. 15; p. 284-309.

Warnecke, H.; Hüser, M. (1995). The Fractal Company – A Revolution in Corporate Culture. Berlin: Springer.

Wolfstetter, Elmar (1996). "Auctions: An introduction." in Journal of Economic Surveys; Vol. 10(4); p. 367–420.

White, S.R.; Hanson, J.E.; Whalley, I.; Chess, D.M.; Kephart, J.O. (2004)."An architectural approach to autonomic computing" in Proceedings of IEEE first international conference on autonomic computing; p. 2–9; New York.

Ygge, Fredrik; Akkermans, Hans; Andersson, Arne (1996). "A Multi-Commodity Market Approach to Power Load Management" in Proceedings of the International Conference on Multi Agent Systems.

Zou, Youyong; Finin, Tim; Ding, Li; Chen, Harry; Pan, Rong (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; Vol. 50; p. 95–101.

Zlot, Robert (2004). Complex Task Allocation for Multirobot Coordination. A thesis proposal.